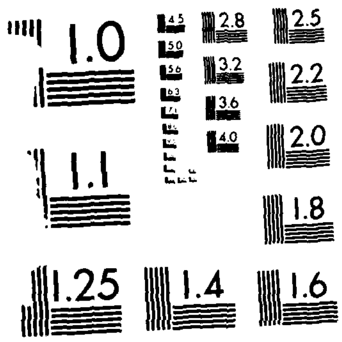
AD-A165 232 IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM RELIABILITY VOLUME 2 PROCEDU.. (U) MARTIN MARIETTA AEROSPACE ORLANDO FL E C SOISTMAN ET AL. DEC 85 1/1

UNCLASSIFIED OR-18173-1 RADC-TR-85-228-VOL-2 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

AD-A165 232

RADC-TR-85-228, Vol II (of two)
Final Technical Report
December 1985

IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM RELIABILITY Procedures for Use of Methodology

Martin Marietta Orlando Aerospace

Edward C. Soistman and Katherine B. Ragsdale

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-85-228, Vol II (of two) has been reviewed and is approved for publication.

APPROVED: Eugene Fiorentino

EUGENE FIORENTINO
Project Engineer

APPROVED: W S Tuthill

W. S. TUTHILL, COLONEL, USAF
Chief, Reliability & Compatibility Division

FOR THE COMMANDER: John A. Ritz

JOHN A RITZ
Acting Chief, Plans Office

AD-A165 232

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) OR 18,173-1	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-85-228, Vol II (of two)
6a. NAME OF PERFORMING ORGANIZATION Martin Marietta Orlando Aerospace 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (RBET)
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 5837 Orlando FL 32855	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) RBET	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-83-C-0050
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62702F PROJECT NO. 2338 TASK NO. 02 WORK UNIT ACCESSION NO. 96

11. TITLE (Include Security Classification)
IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM RELIABILITY Procedures for Use of Methodology

12. PERSONAL AUTHOR(S)
Edward C. Soistman and Katherine B. Ragsdale

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM Mar 83 TO Jan 85	December 1985	104

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Reliability
14	4		Software Quality
9	2		Hardware/Software Reliability Prediction

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The objective of this study was to develop techniques, for predicting total system reliability, which include the combined effects of software and hardware. Since hardware reliability techniques are much further developed, the study emphasized methods of characterizing software reliability. The software reliability prediction methodology contained in the report is compatible with hardware reliability techniques and definitions and is applicable during early development so that the predictions can influence the design and development process.

The software reliability prediction techniques use both software product and development process characteristics to develop estimates of the reliability of the various software components which comprise the system. The software component reliabilities are combined via a Markov model to obtain estimates of software system reliability. Estimates of the

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Eugene Fiorentino	22b. TELEPHONE (Include Area Code) (315) 330-3476 22c. OFFICE SYMBOL RADC (RBET)

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

execution frequencies of the various software components, as a function of the mission profile, are required by the methodology.

Procedures for application of the techniques are provided and are intended for use by a reliability engineer having a basic knowledge of software engineering practices. The techniques offer a rudimentary framework for predicting total system ~~(HW & SW)~~ reliability. Validation and refinement of the techniques using software development and field reliability performance data remains to be accomplished.

17. COSATI CODES (Continued)

Field	Group
18	03

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 PROCEDURE FOR UTILIZATION OF THE COMBINED HARDWARE/SOFTWARE RELIABILITY PREDICTION METHODOLOGY

1.1 Overview and Objectives

The prediction methodology described below was developed by Martin Marietta Aerospace as part of a Research and Technology contract performed for the Rome Air Development Center (RADC).

Although many models exist for measuring and estimating total system reliability, very little is available to assist a procuring agency or software developer in predicting system reliability during the early phases of the Development Life Cycle. Specifically, there has been no software equivalent to MIL-HDBK-217D. Quantitative evaluation of software reliability follows two extremes.

At one end of the spectrum, research has involved the psychological aspects of computer programming by considering the mental processes performed during software creation. Although this basic research is very interesting and may eventually lead to very sophisticated automatic program generators, it is too detailed and imprecise to be usable by a system planner or analyst.

At the other end of the spectrum, many mathematical models are available to measure and estimate software reliability based on historical failure information. These models are extremely valuable, but not until a system is actually developed. They are of little value to a planner who must be able to predict reliability based on the nature of the mission and the intended method of development.

The methodology described here was developed specifically to fill the void. Although it lacks the academic generality of psychological approaches and the statistical accuracy of measurement and estimation methods, it provides a prediction tool that can be used early in the life cycle of software development; a time when alternate approaches can be evaluated and cost tradeoffs can be performed. Specific features of the methodology include:

1. Applicable during early design/development
2. Applicable throughout the development cycle
3. Yields quantitative reliability predictions
4. Utility as a design and process evaluation tool
5. Uses the operational mission scenario as a basis for prediction
6. Compatible with MIL-HDBK-217D techniques and reliability definitions.

1.2 Approach

The methodology was developed to maximize the parallelism of software reliability prediction and classical hardware techniques (Figure 1).

Although the sequence of steps for performing a software reliability prediction are nearly identical to those for a hardware analysis, the terminology, procedures and techniques differ quite extensively.

Whereas hardware components fall into some generic class based on their electrical construction and material composition software components are categorized by their logical makeup and purpose. Like hardware, software has intrinsic characteristics that can generally be used to classify it and obtain a starting or base reliability. Generally, these characteristics are identifiable by the functional requirements imposed on the software before it is even designed. For example, a software component that is required to operate in real time tends to be more error prone than one that operates in a batch environment.

Hardware component reliabilities are adjusted by pi factors determined by the various aspects of their development and operational environments. Factors such as stress, environment, quality, temperature, and technology must be identified and applied to component reliabilities before they can be incorporated into the reliability model. Fortunately, MIL-HDBK-217D provides extensive lists of factors for virtually all situations.

Software component reliabilities are similarly adjusted based on the environment in which the software is developed. Software is immune to stresses as used in hardware analysis, such as temperature and vibration. Software is pure logic, with no physical nature. Its reliability is affected by the characteristics of the software product which predisposed it to error and the manner in which errors are avoided or detected and corrected. These are the only pi factors of concern to software reliability prediction. Unfortunately, extensive lists of factor data similar to that provided by MIL-HDBK-217D for hardware are not available. The appendices to this document present factors that were derived from a relatively large survey of software experts. Statistically, the survey data provides an accurate representation of what practitioners believe the factors to be. Currently, it is the best available data. Although the data available at present is not based on extensive analyses or experimentation, the methodology is sound. As more precise data becomes available, the factors and their values may require revision. However, the method should remain intact. Even with imperfect data, the method will provide relative figures of merit for the analyst to compare alternate approaches to software development. The methodology and the data provide a framework for the creation of a software handbook similar to MIL-HDBK-217D.

MIL-STD-785B prescribes the manner in which hardware reliability block diagrams are to be resolved. Methods of combining serial components,

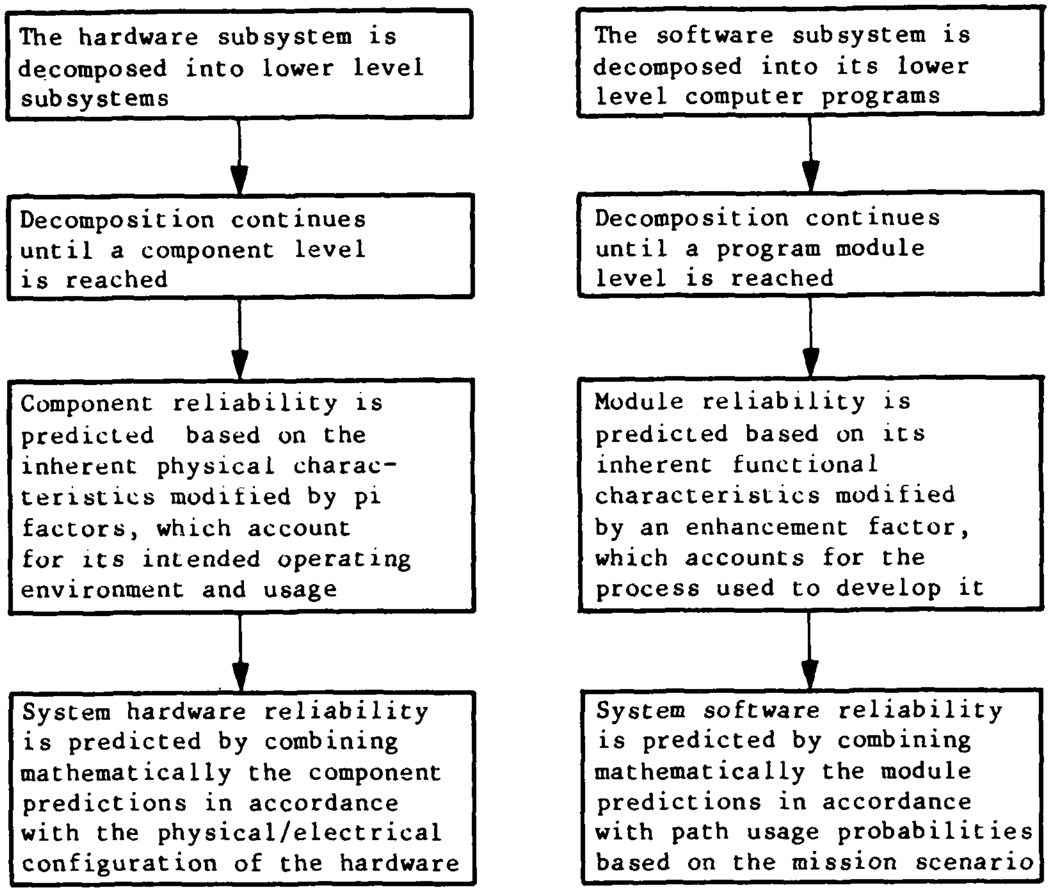


Figure 1. Parallel Hardware and Software Techniques

parallel components, redundant components, etc. have been well developed and practiced by hardware reliability engineers for some time. With few exceptions, classical combinatorial techniques used for hardware analysis. are not usable for software analysis. A logical path through a computer program is a straightforward serial arrangement of critical components. If there were only one path through a computer program, its reliability would simply be the product of the reliabilities of each software component in that path. However, there are typically thousands, and often a nearly infinite number of paths possible. The reliability contribution of each software component becomes a conditional probability. It is the probability that the component will operate successfully, given that it is executed in a particular path. Obviously, the computational aspects of evaluating thousands, or millions, of conditional probabilities are not feasible. Fortunately, a mathematical technique is available to alleviate this situation.

Determining system reliability from the computed hardware and software reliabilities is again slightly different from normal hardware analysis. Software reliability is not directly a function of time, but rather a function of the particular mission scenario that it is required to perform. Whereas hardware is usually considered to have a constant failure rate for a particular mission and environment, software can be considered to have a fixed reliability over a given mission duration and input environment. To determine system reliability, we must multiply software and hardware reliabilities.

Lastly, it must be again emphasized that software reliability is critically mission dependent. Although individual software component reliabilities will not change, their execution frequencies will change for different missions. Whenever a different mission scenario is to be evaluated, the software subsystem reliability must be recomputed.

1.3 Procedure

Application of the reliability prediction methodology is relatively simple. The steps are performed in sequence using tools common to software engineering. Since the methodology does not impose any restriction or special conditions on standard hardware prediction techniques, the classical methods of MIL-STD-785B and MIL-HDBK-217D can be separately applied, without alteration, for all hardware components of the system under analysis. Software reliability prediction is accomplished as described in the following paragraphs.

1.3.1 Preliminary Analysis

Before using the computational aspects of the methodology, it is essential that preliminary analysis of the software system be accomplished. The following are typical software engineering activities that are accomplished during the preliminary design phase of a software development effort. They are considered to be good engineering practices independent of their usage for reliability prediction.

Perform A Functional Decomposition (Appendix A)

During preliminary design of a software product, it is necessary to allocate every functional requirement of the software subsystem to the identifiable and separable components of the total software product. Among the products of this phase is a list of software components (e.g., modules) and their individual functional requirements. Analysis cannot proceed until this step is completed.

Construct A Functional Flow Diagram (Appendix B)

After all components of the software and their individual requirements have been identified, it is necessary to show their functional relationships to each other. A functional flow diagram is a tool that can be used for this purpose. Other tools are also available and may be used in the analysis. The important output of this phase of the analysis is to clearly understand how components pass control to one another. Results of this analysis will be used directly in latter phases of the reliability prediction.

Perform A Mission Thread Analysis (Appendix C)

When the functional flow diagram just described is completed, it is necessary to assign path probabilities to each decision point depicted. For a given mission, each individual branch must be evaluated to determine (or estimate) its probability of execution, given that control has reached the branch point. During the preliminary design phase of the development effort, these values will probably be rough estimates. As the design becomes better developed and understood, the data can be updated to afford greater precision. For real-time applications, these values are critical to the engineering process itself and should be available early in the design phase.

Identify The Individual Component Characteristics (Appendix D)

The reliability of each software component is predicted based on its inherent characteristics (attributes of the software PRODUCT) and its developmental characteristics (attributes of the software development PROCESS).

The functional requirements for each component are known to the analyst as a result of the functional decomposition described earlier (Appendix A). The inherent characteristics used within the prediction methodology are defined in Appendix D and included in the worksheets (Appendix I). The analyst must identify the appropriate characteristics for each software component within the overall software subsystem by choosing those which best describe the functional requirements. By using the worksheets provided, the analyst can determine the types of errors most likely to be encountered during the development process.

The techniques to be used during software development must also be identified at this time in the analysis. The techniques can be classified as being either an error avoidance technique or an error detection technique. The techniques used within the prediction methodology are also defined in Appendix D and included in the worksheets (Appendix I). Typically, the techniques to be employed (structured approaches, independent test, etc.) are identified in a Software Development Plan or equivalent document. Such a plan is considered to be good engineering practice and is usually a required, deliverable data item for DoD contracts. By using the worksheets, the analyst can determine the collective effectiveness of the planned techniques for the type of errors expected.

1.3.2 Calculate the Software Reliability

Compute Individual Component Reliabilities (Appendix E)

Individual reliabilities are computed in accordance with the methods derived and described in Volume I of this report. The technique is described briefly in Appendix E. Worksheets have been devised to simplify the computation process. Having already gathered the individual component characteristic information just described, the analyst simply performs the operations described on the worksheets. The only computations involved are simple addition, multiplication, and division, which can be performed either manually or with a small calculator.

Compute Overall Software Reliability (Appendix F)

After the individual component reliabilities have been calculated and the path probabilities (described under functional thread analysis) are known, the analyst must construct a transformation matrix as described in Appendix F. The matrix is actually a representation of all of the joint probabilities of a component successfully executing and passing control to the next component. Calculation of the overall software reliability involves the inversion of this matrix and could involve relatively complex mathematics. Although matrix inversion can be accomplished manually for software containing relatively few functional components, it is highly desirable to computerize this step of the analysis.

1.4 Examples

Two examples have been included to demonstrate the methodology application.

1.4.1 Detection and Warning System (Appendix G)

This example was created specifically for demonstration purposes. It describes a typical, albeit simplified, software application. The software receives target data from some sensor. If the data meets certain correlation criteria, an acquisition mode is initiated and the target is tracked assuming that the acquisition was successful. If the tracked target is hostile, a report is generated. In any event, the system returns to the search mode of operation.

1.4.2 Assault Breaker (Appendix H)

This example was taken directly from a completed Martin Marietta software development project. Assault Breaker software was responsible for guidance and control of a tactical missile. The software was required to operate in real time with relatively precise timing constraints. It was developed using modern top-down, structured design approaches and was thoroughly documented during its development. This software was chosen as an example because it demonstrates the effectiveness of some software techniques and represents a good example of the manner in which path execution probabilities can be determined.

1.5 Summary

The methodology presented here provides a workable technique for determining software reliability information at any stage of software development, subsequent to preliminary design. It can and should be used recursively throughout the development process to evaluate the approaches being taken, to alert management when reliability predictions fall short of required performance criteria, and to evaluate the impact of alternate approaches.

Appendix I includes copies of the worksheets required for analysis.

APPENDIX A

FUNCTIONAL DECOMPOSITION

1.0 INTRODUCTION

Decomposition of computer software is accomplished much like hardware. Whereas hardware subsystems can be segmented wherever a connection has been (or will be) made, software can be broken anywhere in the sequence of commands that it executes. In both cases, however, it is illogical to disconnect components except at the physical (or logical) boundaries of complete subunits. For hardware we might decompose a system into black boxes, decompose the boxes into printed circuit boards, decompose the boards into circuits and finally decompose the circuits into their respective electrical components. It is essential that every phase of the process yields complete subunits. Computer programs are similarly decomposed.

Figure A-1 illustrates the generally accepted terminology associated with software decomposition. It lists the terminology specified in the proposed military standard DoD-STD-2167 and has been extended to the lowest possible level by this author. At the highest level, software is defined as a configuration item, one which is defined by and for the procuring agency. It has considerable contractual significance but has no logical or functional characteristic. At the other end of the spectrum, the level of detail is so specific that prediction is not possible until after implementation.

Although the software prediction methodology is not affected by the level to which the software is decomposed, it is practical to define its applicability as ranging from CSC level through the module level. Generally, CSC level decomposition is possible during the requirements definition phase of software development, unit decomposition is possible during preliminary design, and module decomposition is possible during the detailed design phase. At each milestone the software reliability prediction methodology can be re-applied with greater accuracy. For generality, the term "software component" is used to include all levels of software decomposition between the CSC and module level.

- COMPUTER SOFTWARE CONFIGURATION ITEM (CSCI) - software aggregate which is designated by the procuring agency for configuration control
  - COMPUTER SOFTWARE COMPONENT (CSC) - a functional or logically distinct part of a computer software configuration item
    - UNIT - the lowest level logical entity specified in the detailed design which completely describes a non-divisible function in sufficient detail to allow implementing code to be produced and tested independently of other units
      - MODULE - the lowest physical entity specified in the detailed design which may be assembled or compiled alone
        - INSTRUCTION - a single line of code which may correspond to a single action of the computer or may be automatically translated into a series of single actions of the computer
          - OPERATION - the action to be performed by the computer
          - OPERANDS - the symbolic or absolute addresses of the computer memory where the data to be processed reside.

Figure A-1. <u>Software Decomposition Process and Terminology</u>

2.0 METHODOLOGY

The analyst must begin his reliability prediction with a relatively complete description of the overall software functional requirements and the manner in which these requirements are allocated to the various sub-units that make up the software system. Such a functional decomposition is required for the software development team as well as the reliability engineer. Typically, a software project manager will segment the overall job into subtasks which can be individually assigned, tracked, tested and eventually integrated back into the overall software system. It is considered good engineering practice to perform this segmentation functionally. That is, top-down, structured design approaches provide both the mechanism and the motivation for the accomplishment of a functional decomposition very early in the design effort. The reliability engineer should obtain this top-level breakout as soon as it becomes available.

3.0 USE OF THE FUNCTIONAL DECOMPOSITION

Properly prepared design documentation should include identification of all components (modules), their functional requirements, their inputs and outputs, performance requirements, testing plans, development schedule and their interface requirement. When this information is available, each module can be evaluated with respect to its contribution to the overall reliability prediction. The functional flow diagram, described in Appendix B, and the component reliability worksheets can be prepared.

APPENDIX B

FUNCTIONAL FLOW DIAGRAM

1.0 INTRODUCTION

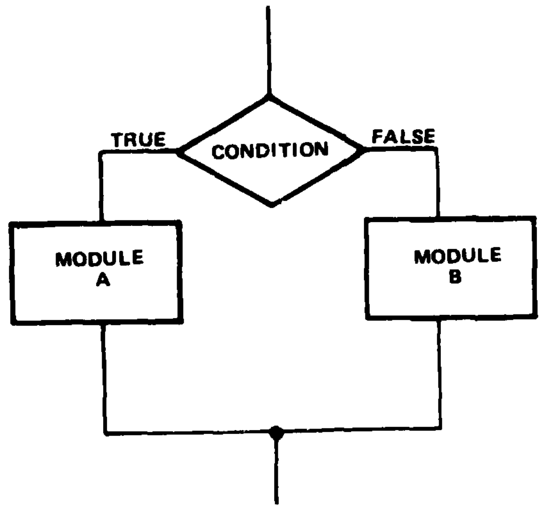
The functional flow diagram is an integral part of the software reliability prediction methodology. Although the particular form of the diagram is insignificant, the data it contains is critical.

The most commonly used tool today is the flowchart. Both of the example problems included in this report (Appendices G and H) use flowcharts to depict the functional flow of control within a software system.

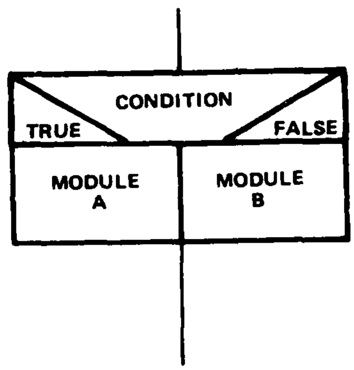
Another method commonly used is the Visual Control Logic Representation (VCLR) diagram. These diagrams have the added feature of directly supporting structured programming. Each symbol used is completely self-contained and has a single entry and a single exit. The VCLR is an equally sufficient tool for use in identifying the data required for the prediction methodology. Paths are a little less obvious and determining path probabilities may be a bit more difficult than when working with flowcharts, but the increase in structuredness should be worth the tradeoff.

It is very likely that diagrams as such will be used less and less frequently as Program Design Languages (PDLs) become more widely used. PDLs are languages used to describe the design. They may or may not produce executable code. Their primary purpose is to specify and create the logic of the computer program. They have the features of being precisely definable and are usually accompanied by a variety of automatic design checking tools (consistency, continuity, etc.). Ada can, and probably will, be used as a PDL. As its usage spreads, we may enter a programming environment where the software design as well as the code is expressed in machine readable form compiled and checked. This could greatly enhance the application of the software reliability prediction methodology. Actual frequency counts of logic segments can be produced by the PDL compiler. That is, the actual path probabilities could be automatically and continuously monitored during the design phase. Automatic computation of software reliability would become feasible.

Figure B-1 illustrates equivalent logic representations using a flowchart, a VCLR, and a PDL, respectively.



FLOW CHART



VCLR

```
IF    CONDITION THEN
      MODULE A
ELSE
      MODULE B
ENDIF
```

PDL

Figure B-1. Equivalent Logic Representations

2.0 USE OF FUNCTIONAL FLOW DIAGRAMS

The reliability engineer requires detailed information about the logical relationships between individual components of the software system. Regardless of the format of the diagrams, he should be able to follow the flow from one component to the next through the program. He should likewise be able to identify mission threads through the program. That is, if the software is required to perform more than one mission, the conditions which define the mission, should be self-evident within the flow diagram. Similarly, a single mission application may be required to progress through several modes of phases (e.g., boost, burnout, ballistic, final). The reliability engineer should be able to identify the criteria and logic involved based on the information contained in the flow diagram.

A thorough and accurately prepared functional flow diagram will provide the analyst with the information required to accomplish the next step of the analysis (Thread Analysis).

APPENDIX C

MISSION THREAD ANALYSIS

1.0 INTRODUCTION

A clear understanding of the mission requirements of a computer program is essential to the successful application of the reliability prediction methodology. Software failure mechanisms are not the same as those which influence hardware reliability. Whereas hardware is affected by environmental and stress conditions, software is not, at least not in the same sense. The environment in which software operates consists only of the internal state of its storage and registers and the external influences it sees via data coming into or leaving its storage. Software is stressed when accumulated internal and external effects cause it to execute a logic path that has never been traversed before or when it follows a previously used path with an internal state which has never occurred on that path before. A computer program which continuously repeats the same logic paths with the same data will either fail on the first pass or will never fail. Unfortunately, even a simple program has an extremely large (possible infinite) number of logical paths through it. Equally awesome is the size of the input domain even when only a few 16-bit variables are required.

For a given path and a given state, the reliability is, in fact, deterministic; i.e., it will either work or it won't. However, historical software performance data invariably shows that software failures occur probabilistically. The only explanation possible is that the software experiences state changes probabilistically. Exhaustive testing to check every possible state of every possible logic path is highly impractical and typically impossible. The reliability prediction methodology described in this report relies on a statistical technique to account for all possible paths by use of mission scenarios which are defined by path probabilities at every decision point in the software. These probabilities are assigned based on the functional flow diagram (Appendix B) and the construction of a mission profile based on software mission requirements.

2.0 MISSION THREAD ANALYSIS

Branch points within the flow of a computer program must be expressed in machine-understandable form. For the decision to be properly implemented, it is essential that the programmer understand the exact conditions under which the branch will occur. Appendix H describes the process used in determining what path probabilities to assign to various branches based on an engineering description of the software design. There is no set formula for how to assign these probabilities, so the analyst must thoroughly understand the mission requirements and apply that knowledge in making sound estimates. In one case, the values might be precisely determined through mathematical interpretation of the stated requirement such as the Assault Breaker example in Appendix H. In another case, they may involve engineering judgements. The example depicted in Appendix G includes both. In one case, the branch criteria was based on a required time period for preventative maintenance and the path probabilities could be directly computed. In another case, the path probability assigned to leave the search mode was based on an estimate that the system would spend 90 percent of its time in the search mode.

The best recommendation possible at this point is to suggest that several analyses be performed using different path probabilities at those branches where there is ambiguity. By varying the values, one at a time, the analyst can determine the sensitivity of the reliability prediction to the value in question.

1.0 INDIVIDUAL COMPONENT CHARACTERISTICS

This appendix presents the lists of characteristics which were identified during the study as having a significant influence on software reliability. Tables D-1 through D-3, respectively, list:

- o Inherent Characteristics of the software PRODUCT which influence its error proneness

- o Error Avoidance Characteristics of the PROCESS used to develop the software

- o Error Detection Characteristics of the PROCESS used to develop the software.

The lists are ordered in the same sequence that they appear in the worksheets (Appendix I) so that the analyst can refer to the definitions while determining which characteristics best describe the software and development process for which a reliability prediction is being made.

TABLE D-1 Inherent Characteristics.

OPERATIONAL APPLICATION TYPE - This characteristic is used to describe the predominant use of a software component. For example, if the purpose of the module is to issue commands to hardware components, we would say the module is of the "predominantly controlled type". even though it includes computational commands.

CONTROL - The action of initiating, sequencing, terminating or otherwise influencing the operation of system components external to the software.

REAL-TIME - The processing of information or data in a manner sufficiently rapid that the results of the processing are available in time to influence the process being monitored or controlled.

INTERACTIVE - A method of conversational input/output wherein the software produces an output which invokes a responsive input or receives an input which requires a responsive output.

COMPUTATIONAL - The process wherein internally available data is combined, rearranged and/or otherwise manipulated to alter its state. For example a module whose purpose is to convert measurements from one dimension to another should be regarded as being computational.

MISSION VARIABILITY - In most large scale software applications, a variety of missions or modes of operation are supported. For example, software requirements for embedded software in a missile system may involve distinct modes of operation such as pre-flight, boost and ballistic activities. Some modules will perform the same activities regardless of the mission type, while others will have distinctly different characteristics depending on the mission mode. MANY and SEVERAL operational missions are relative terms that may be interpreted at the discretion of the reader.

FUNCTIONAL COMPLEXITY - In order to meet its intended purpose, a module may be required to perform more than one specific task. These entries accommodate the fact that some functions are relatively easy to design and code whereas others can require extensive and highly complex logic.

SYSTEM INTERACTION - This category is a refinement of earlier categories. Interface requirements are as previously defined. EXTENSIVE and MINIMAL are relative terms that may be interpreted by the analyst.

INPUT DOMAIN VARIABILITY - This category is a refinement of earlier categories. Here, the interest is not in the quantity of inputs required, but rather the domain from which it comes. For example, a function which requires yes or no answers to many questions would have a NARROW RANGE of values (yes or no). On the other hand, a single input of an angle measurement might have a domain of -180.0000 to +180.0000 degrees. This one would be considered to have a WIDE RANGE of inputs.

TABLE D-1. Inherent Characteristics (Cont).

ERROR-PRONE/ERROR-FREE - These adjectives are used to distinguish the effects on module reliability caused by the SOURCE of data inputs. A device which contains self-checking features to ensure that its inputs to the computer are correct would be considered error-free. On the other hand, other input devices, such as human operators, may be considered to be error-prone.

TABLE D-2. Error Avoidance Characteristics.

QUALITY ASSURANCE ORGANIZATION - A group responsible for the planned and systematic review of the software development process and its products to provide adequate confidence that the item or product conforms to established technical requirements.

TEST ORGANIZATION - A group responsible for preparing test plans and procedures, executing the test procedures, and analyzing the test results in order to verify that the system performed its intended functions. This group is also responsible for documenting problems detected during testing and verifying by retest that corrections to the software work properly.

INDEPENDENT VERIFICATION AND VALIDATION (IV&V) - Verification and validation of a software product performed by an organization that is both technically and managerially separate from the organization responsible for developing the product.

SOFTWARE SUPPORT LIBRARY - A software library containing computer readable and human readable information relevant to a software development effort.

CONFIGURATION CONTROL BOARD - The authority responsible for evaluating and approving or disapproving proposed engineering changes, and ensuring implementation of the approved changes.

SOFTWARE DEVELOPMENT PLAN - This document presents the comprehensive plan for the project's software development activities by describing the software development organization, the software design and testing approach, the programs and documentation that will be produced, software milestones and schedules, and the allocation of development resources.

SYSTEM REQUIREMENTS SPECIFICATION - This document states the technical and mission requirements for a system as an entity, allocates requirements to functional areas, and defines the interfaces between or among the functional areas.

INTERFACE DESIGN SPECIFICATION - This is an optional document which is required whenever the system contains two or more computers that must communicate with each other. It provides a detailed logical description of all data units, messages, control signals and communication conventions between the digital processors.

SOFTWARE REQUIREMENTS SPECIFICATION - This document establishes the requirements for the performance, design, test and qualification of the computer program.

TABLE D-2. Error Avoidance Characteristics (Cont).

SOFTWARE FUNCTIONAL DESIGN SPECIFICATION - This document establishes the functional design of the software at the computer program level. It provides sufficient design information to accomplish the goals of the Preliminary Design Review.

SOFTWARE DETAILED DESIGN SPECIFICATION - This document provides complete programming design sufficiently detailed for a programmer to code from with minimal additional direction.

REQUIREMENTS TRACEABILITY MATRIX - A set of tables which provides traceability of software requirements from the system specification to the individual item requirements specifications, to the design specification which implements the requirements, and to the software plans and procedures that verify that requirements have been fully implemented.

STRUCTURED ANALYSIS TOOLS - These define a systematic method of using function networks and other tools to develop an analysis-phase model of a system. Typical tools include Data Flow Diagrams, Data Dictionaries and structured English.

PROGRAM SPECIFICATION LANGUAGE (PSL) - A language used to specify the requirements, design, behavior, or other characteristics of a system or system component.

PROGRAM DESIGN LANGUAGE (PDL) - A language with special constructs and, sometimes, verification protocols used to develop, analyze, and document a design.

HIGH ORDER LANGUAGE (HOL) - A programming language which provides compression of computer instructions such that one program statement represents many machine language instructions. It is non-problem specific and is used by programmers to communicate with the computer.

HIERARCHICAL DESIGN - A design which consists of multiple levels of decomposition, general to specific. It is a structured approach with the additional restriction that program control is accomplished hierarchically. That is, program modules may only invoke other modules which are subordinate to them.

TOP-DOWN DESIGN - An ordering to the sequence of decisions which are made in the decomposition of a software system, by beginning with a simple description of the entire process (top level). Through a succession of refinements of what has been defined at each level, lower levels are specified.

STRUCTURED DESIGN - A disciplined approach to software design which adheres to a specified set of rules based on principles such as top-down design, modularization, stepwise refinement, etc.

TABLE D-2. Error Avoidance Characteristics (Cont).

SINGLE FUNCTION MODULARIZATION - An organization of the functions of the computer program into a set of discrete program modules each of which is designed to perform a single function.

STRUCTURED CODE - A code that has been generated with a limited number of well-defined control structures using stepwise refinement.

AUTOMATIC MEASUREMENT TOOLS - This category includes all computer programs which evaluate other computer programs. They may be used to verify compliance with coding standards, to measure progress, or to provide a measure of complexity. They may be applied to any or all phases of the development cycle.

AUTOMATIC TEST TOOLS - This category includes all computer programs that automatically devise and/or execute tests on other computer programs by analysis of the path logic and variable domains of the software being test d and construction of test data sets which will exercise all logical paths under all or extreme input conditions.

TABLE D-3. Error Detection Characteristics.

FREQUENT/INFREQUENT - These are relative terms that may be interpreted by the reader. In general, however, it is preferred that "frequent" be used to describe activities that occur on a regular, scheduled basis (e.g., weekly). "Infrequent" carries the connotation that the activity is less rigidly planned and accomplished (e.g., whenever a problem is suspected).

WALKTHROUGH - A review process in which an analyst, designer or programmer leads one or more peers through a segment of the software product which he or she has developed.

PROGRESS REVIEW - For purposes of this survey, a progress review is a periodic report given to an individual's supervisor to provide an assessment of the state of completion of a software product. This is in contrast to a walkthrough which is conducted among peers and is primarily technical in nature.

QUALITY AUDIT - For purposes of this survey, a quality audit is an announced or unannounced inspection of a software product or process. For example, an audit may consist of an inspection of a portion of a programmer's code to verify compliance with programming standards.

SOFTWARE PROBLEM REPORT - A report of a program deficiency identified during software qualification, test, system integration and test, or system operation, which appears to be software related.

SPECIFICATION CHANGE NOTICE - A formal notification of a change in the specification.

ENGINEERING CHANGE NOTICE - A document used to process changes to baseline documents and which includes both notice of an engineering change to a configuration item and the supporting documentation by which the change is described.

SOFTWARE REQUIREMENTS REVIEW (SRR) - A review to achieve formal agreement between the customer and the developer that the software requirements specifications are complete and accurate.

PRELIMINARY DESIGN REVIEW (PDR) - A formal technical review of the basic design approach. It is held after the completion of preliminary design efforts but prior to the start of detailed design. See also SYSTEM DESIGN REVIEW and CRITICAL DESIGN REVIEW.

CRITICAL DESIGN REVIEW (CDR) - A formal technical design review conducted to ensure that the detailed design correctly and completely satisfies the requirements. It is conducted after completion of the detailed design but prior to coding. It establishes the design baseline.

TABLE D-3. Error Detection Characteristics (Cont).

TEST READINESS REVIEW (TRR) - A review conducted prior to each test to ensure adequacy of the documentation and to establish a configuration baseline.

FUNCTIONAL CONFIGURATION AUDIT (FCA) - Audit to verify that the actual performance of the configuration items complies with the B-5 development specifications.

PHYSICAL CONFIGURATION AUDIT (PCA) - A formal examination of the as-built version of a configuration item against its technical documentation to ensure the adequacy, completeness, and accuracy of the technical design documentation.

UNIT LEVEL TESTING - Testing to verify program unit logic, computational adequacy, data handling capability, interfaces and design extremes, and to execute and verify every branch.

PRELIMINARY QUALIFICATION TESTING (PQT) - An incremental testing process which provides visibility and control of the computer program development during the time period between the Critical Design Review (CDR) and Formal Qualification Testing (FQT); conducted for those functions critical to the CPCI.

FORMAL QUALIFICATION TESTING (FQT) - Testing conducted prior to Functional Configuration Audit to demonstrate CPCI compliance with all applicable software specifications.

SOFTWARE INTEGRATION TESTING - Tests of the overall computer program used to verify proper module interfaces with respect to sequencing, timing, and data compatibility.

SYSTEM INTEGRATION TESTING - The process of testing an integrated hardware and software system to verify that the system meets its specified requirements.

OPERATIONAL FIELD TESTING - Test performed by operational personnel in the operational environment. These can be the same tests performed earlier during FQT.

APPENDIX E

INDIVIDUAL COMPONENT RELIABILITIES

1.0 INTRODUCTION

Just as hardware components can be classified into component categories such as resistors, capacitors, and diodes, software can be categorized into characteristic groups. In the case of software, however, the distinction between groups is based on logical composition rather than physical makeup. A direct relationship between the complexity of a computer program and its reliability is intuitively expected. Likewise, it is intuitive that the complexity of the software is related to its intended application (the programmatic complexity of the design is considered later). In other words, even before it is designed or implemented, real time software is expected to be more error prone than batch software where timing is not a critical consideration. Similarly, historical evidence shows that programs with a large amount of interface requirements experience higher failure rates than those that contain minimal interfaces.

To determine individual component reliability (probability of success on a single execution), the analyst must evaluate the characteristics of the component itself as well as the characteristics of the process which will be used to develop it. The methodology described in the appendix is derived and explained in Section 5.0 of Volume I. The worksheets included in Appendix I of this volume were created to simplify the application of the formulas. As was explained earlier, the numeric values were statistically derived from survey data collected during the study.

2.0 METHODOLOGY

2.1 Overview

The methodology for predicting individual component reliability involves the identification or calculation of its inherent reliability and the application of an enhancement factor which is calculated as a function of the development process characteristics discussed in Appendix D. Specifically:

$$R_c = R_i + E(1 - R_i) \tag{1}$$

and,

$$E = \frac{A}{1 - D\ (1-A)} \tag{2}$$

where: R_c is the expected component reliability

R_i is the inherent reliability of either the process or the component

E is the enhancement factor achieved by the application of error avoidance and detection techniques

A is the single factor which describes the effect of applying one or more error avoidance techniques during development

D is the single factor which describes the effect of applying one or more error detection techniques during development.

2.2 Expected Component Reliability

The expected reliability of a software component is a function of the inherent characteristics of the component and the characteristics of the development process used to produce it. Unfortunately, there is insufficient historical data available to isolate, with any degree of confidence, the casual relationship between a specific characteristic or development technique and the resulting effect on component reliability. The term "reliability" is used here to connotate the probability that the software component will perform its intended functions correctly the next time it is executed. That is, component reliability is defined as the probability of success in a single trial. If it were possible to extensively exercise the component in a controlled experiment, its reliability could be approximated by the ratio of its successful executions to its total executions. In fact, if at the end of the development cycle, extensive run data is available for a particular software component, or that component has other dependable failure data, we would bypass all of the following analyses and simply use the known, or measured component reliability. However, prior to its development, we predict component reliability by estimating its inherent reliability and modifying that estimate by an enhancement factor related to the manner in which it will be developed:

$$R_c = R_i + E(1 - R_i) \tag{3}$$

where: R_c is the component reliability (probability of success)

R_i is the inherent reliability of either the process or the component (described in Section 2.3)

E is the enhancement factor achieved by the application of error avoidance and detection techniques (derived in Section 2.4).

2.3 Inherent Component Reliability - R_i

There are several approaches to determine inherent software component reliability, each of which has both advantages and disadvantages. Any measure which presents a ratio of successful implementations to total implementations may be used in the prediction methodology. Some of the more obvious measures are discussed below:

1 Assume that R_i is equal to zero. This causes equation (3) to reduce simply to the enhancement factor. At first glance, this approach appears to be a gross simplification. It essentially says that unless an effort is made to avoid and/or detect errors, the software component will not work. We feel that this is the most theoretically sound approach. However, it assumes that the developer has absolutely no knowledge of the product he is responsible to develop. Even a casual knowledge of what he's supposed to do can be considered an error avoidance technique. This assumption carries with it the additional assumption that the checklist of avoidance and detection techniques is exhaustive. It does, however, define a lower bound on the reliability prediction when the developmental characteristics are known.

2 Assume that the inherent UNreliability is proportional to measured fault densities of existing software which has similar characteristics. This approach has the advantage of data availability. Although there is a fairly wide range of measured values of faults per line of code, there is sufficient historical data for an analyst to make a sound engineering determination of the best figure to use. Care must be taken, however, to distinguish how and when the data used was collected. Many organizations do not begin counting faults until the software is tested in the overall system while others begin recording failure data as soon as individual components have completed unit test. The prediction methodology assumes that R_i includes consideration of all errors made, not just the ones recorded subsequent to integration testing. This method should produce an upper bound on the reliability prediction due to the fact that the actual number of faults in a software product cannot be less than the number recorded.

3 Assume that the inherent UNreliability is proportional to a fault density which has been interpolated from the range of historically recorded fault densities. The interpolation could be based on the same characteristics already discussed in Table D-1. Although such a scheme has not yet been formulated, it is the opinion of the author that one could be created and that it would provide the most unbiased measure.

2.4 Enhancement Factor - E

Figure E-1 illustrates the relationship between inherent reliability, avoidance effectiveness and detection effectiveness. The figure introduces some terminology not previously described:

R_i Inherent reliability.

N Total possible variations implemented (all possible combinations of functions to be performed in all possible input domains).

NG Total variations inherently implemented correctly. These are the variations that would have been properly implemented without process enhancement.

NB Total variations inherently implemented incorrectly. These are candidates to be avoided or detected.

I_i Number of variations being worked on during the i'th iteration. These include the original errors to be eliminated plus reworks of errors discovered on the previous iteration.

NGG Number of variations which "pass thru" the avoidance/detection filters because they are already correctly implemented.

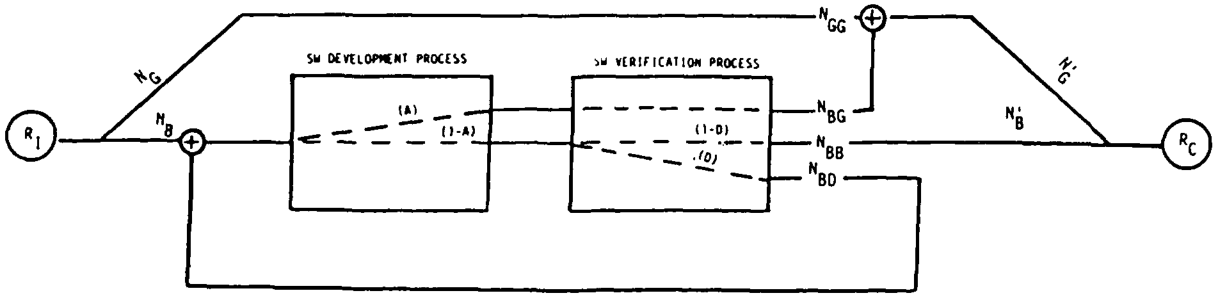
NBG_i Number of previously incorrect implementations which were successfully avoided on the current iteration.

NBB_i Number of previously incorrect implementations which were neither avoided nor detected on the current iteration.

NBD_i Number of previously incorrect implementations which were successfully detected and returned for rework.

A,D These are the error avoidance and detection factors.

The process depicted represents a typical software development operation. As a result of the inherent characteristics of the software to be developed, errors will be made. The development team will attempt to avoid making those errors by the application of software engineering techniques. Recognizing that they will probably not avoid all errors, tests and other detection techniques are implemented to locate and rework the faults. Avoided errors will exit the process as corrected implemented variations. Detected errors will be reworked by the process until they either are avoided or escape the detection mechanisms. Eventually, all N variations exit the process. Since the enhancement factor is an improvement factor, it is defined as:



$R_C = R_I + E(1-R_I)$ where, $E = \dfrac{A}{1 - D(1-A)}$

Figure E-1. Relationship of R(I), R(C), A and D

$$E = \frac{NBG}{NBG + NBB} = \frac{\text{Number of Corrected Bad Implementations}}{\text{Total Number of Bad Implementations}}. \qquad (4)$$

It can be shown (a detailed derivation is presented in Section 5.0 of Volume I) that the quantitative terms N--, including the original number of variations N, cancel out leaving only the error avoidance and detection probabilities. That is, the enhancement factor is simply a function of the PROCESS characteristics:

$$E = \frac{A}{1 - D(1-A)} \qquad (5)$$

where: A and D are the error avoidance and detection factors described in Section 2.4.2.

2.4.1 Effects of Inherent Component Characteristics

Recognizing that error avoidance and detection techniques are not equally effective against all error types, it is desirable to weigh the enhancement factor in accordance with the expected distribution of error types which are inherently expected to occur based on the characteristics of the software component being developed. Table D-1 lists the characteristics which were investigated during the study. The numeric data presented in Appendix I (worksheet No. 1) represents the distribution of errors which can be expected as a result of those characteristics.

If we define C(j) as the percentage of errors of type j to be expected in the module being evaluated, it follows that:

$$C(j) = \sum_{i=1}^{i=N} \frac{c(j,i)}{N} \qquad (6)$$

where: c(j,i) is the percentage of errors of type j caused by inherent characteristic i, (listed in Volume II)

and N is the number of inherent characteristics applicable.

The enhancement factor is, therefore, more accurately defined as:

$$E = \sum_{j=1}^{4} \frac{c(j)A(j)}{1-D(j)(1-A(j))} \qquad (7)$$

where: A(j) and D(j) are the error avoidance and detection factors as determined with respect to error type j.

2.4.2 Error Avoidance and Detection Factors

As described earlier, software development characteristics can be categorized in terms of their contributions to error avoidance or error detection. Virtually any activity during the development life cycle can be evaluated in terms of these two characteristics. The software reliability prediction methodology uses measures of error avoidance and error detection effectiveness which are based on the planned technical and managerial techniques and methods used to develop the software under investigation.

It is generally accepted that certain development techniques are good and will make the software better. For example, it is generally agreed that structured approaches are good and will have a positive influence on the quality and reliability of the product. Quantification of the effects is typically attempted after the development is complete and the results are rarely applicable to new projects. While the approach described herein has a limitation due to the unavailability of detailed historical records, the method is directly applicable to any software development venture.

Error avoidance effectiveness is calculated as the probability of not introducing an error given the opportunity for making the error. Mathematically, it is computed as unity minus the probability that a hypothetical error will not be avoided by any of the techniques employed. If we define A(j) as the probability of avoiding errors of type j, it follows that:

$$A(j) = 1.00 - \prod_{i=1}^{i=N} (1.00 - a(j,i)) \tag{8}$$

where: a(j,i) is the probability of error type j being avoided by the application of technique i, (listed in Volume II)

and N is the number of techniques employed.

Error detection effectiveness is similarly calculated as the probability that an existing error will be discovered and corrected. Again, it is mathematically computed as unity minus the probability that a hypothetical error will not be detected by any of the techniques employed. If we define D(j) as the effectiveness of detecting errors of type j, it follows that:

$$D(j) = 1.00 - \prod_{i=1}^{i=N} (1.00 - d(j,i)) \tag{9}$$

where: d(j,i) is the probability of error type j being detected by the application of technique i, (listed in Volume II)

and N is the number of techniques employed.

3.0 USE OF THE METHODOLOGY

After the inherent and developmental characteristics of the individual software components have been identified (see Appendix D), the calculation of individual reliabilities is relatively simple. Appendix I of this report includes worksheets designed specifically for this purpose. The worksheets are self-contained and may be used directly without understanding of the derivation described above. The numeric values assigned to the factors listed were derived from the results of the survey performed as a part of this study.

APPENDIX F

OVERALL SOFTWARE RELIABILITY

1.0 INTRODUCTION

This Appendix describes and derives the technique used by the reliability prediction methodology. It assumes that the analyst has already identified the software components (modules) that make up the software subsystem, that their individual reliabilities have been calculated, and that a mission thread analysis has produced inter-module path probabilities. The method described utilizes mathematical methods usually associated with a Markov process including a matrix inversion technique.

The rationale for this approach is based on the recognition that software reliability is not only a function of the reliabilities of its collective modules but also a function of the execution sequence(s) of those modules. The probability of a module failing is a conditional probability that it will fail GIVEN that it was executed. This is essentially a duty cycling effect.

Hardware duty cycling effects are generally considered in reliability predictions at the highest level only. Individual components within a circuit are all operational or they are all dormant. Since each component is essential to the continuity of the circuit, duty cycling effects have no meaning at this level and can be ignored.

Duty cycling effects, however, are probably the most critical determinant of software reliability. Software cannot fail unless it is being utilized. Its non-operating reliability is one. A logic path does not exist during a time period (albeit microseconds) when it is not being utilized. The functional flow of control through a computer program is, in fact, its reliability block diagram.

2.0 MARKOV PROCESS

It is possible to predict overall software subsystem reliability by combining the individual module reliabilities in accordance with their expected usage during the application or mission under analysis. This is accomplished by use of a Markov process as suggested by Cheung [1]. The approach is based on the fact that individual software components contribute to the overall reliability when, and only when, they are executed.

The flow of control between components of a software program can be considered a Markov process if we assume that the component reliabilities are independent. Suppose a given software program has n components. It is necessary to know the reliability of each component and the probability of going from one component to another. The component reliabilities are in the diagonal matrix R and the path probabilities are in the matrix P; i.e.,

$$R = \begin{bmatrix} R_1 & 0 & 0 & \cdots & 0 \\ 0 & R_2 & 0 & \cdots & 0 \\ 0 & 0 & R_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & R_n \end{bmatrix} \qquad P = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ P_{31} & P_{32} & \cdots & P_{3n} \\ \vdots & \vdots & & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{bmatrix}$$

where R(i) is the reliability of component i and P(ij) is the probability that control is passed from component i to component j.

The matrix Q is the product of the matrices R and P. The ij'th entry represents the joint probability that component i will execute correctly (R(i)) AND pass control to component j (P(ij)).

$$Q = \begin{bmatrix} R_1*P_{11} & R_1*P_{12} & \cdots & R_1*P_{1n} \\ R_2*P_{21} & R_2*P_{22} & \cdots & R_2*P_{2n} \\ \vdots & \vdots & & \vdots \\ R_n*P_{n1} & R_n*P_{n2} & \cdots & R_n*P_{nn} \end{bmatrix}$$

By considering each component to be a state and by defining two additional states, C and F, for correct program termination and failed termination of the program, respectively, a Markov chain can be constructed with n+2 states. The transition matrix, T, is formed by adding two rows and two columns to the matrix Q. The additional two rows and columns are for the states C and F. The matrix T is defined as follows:

$$
T = \begin{array}{c} \\ 1 \\ 2 \\ \cdot \\ \cdot \\ n \\ C \\ F \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} 1 & 2 & & & & n & C & F \end{array} \\
\begin{bmatrix}
R_1*P_{11} & R_1*P_{12} & . & . & . & R_1*P_{1n} & 0 & 1-R_1 \\
R_2*P_{21} & R_2*P_{22} & . & . & . & R_2*P_{2n} & 0 & 1-R_2 \\
. & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . \\
R_n*P_{n1} & R_n*P_{n2} & . & . & . & R_n*P_{nn} & R_n & 1-R_n \\
0 & 0 & . & . & . & 0 & 1 & 0 \\
0 & 0 & . & . & . & 0 & 0 & 1
\end{bmatrix}
\end{array}
$$

The ij'th entry of T is the probabilty of going from state i to state j in one step. The ij'th entry of T*T is the probability of going from state i to state j in two steps. The ij'th entry of T*T*T is the probability of going from i to j in three steps. The reliability of the software is the probability of going from state 1 to state C in x steps or less as x approaches infinity. Thus, to compute this reliability, we would need to calculate

$$\sum_{i=1}^{i=x} T^i, \text{ as x approaches infinity.}$$

There is a simpler way of computing the reliability of a program using the matrix Q. Let

$$S = I + Q + Q^2 + Q^3 + Q^4 + \ldots$$

where I is the identify matrix.

Note that:

$$
\begin{aligned}
(I - Q) * (I + Q + Q^2 + Q^3 + Q^4 + \ldots) &= I + Q + Q^2 + Q^3 + \ldots \\
&\quad - Q - Q^2 - Q^3 - \ldots \\
&= I,
\end{aligned}
$$

and so,

$$I + Q + Q^2 + Q^3 + Q^4 + \ldots = (I - Q)^{-1}.$$

It follows that,

$$S = (I - Q)^{-1}.$$

Letting S(1n) be the entry in the first row and n'th column of S, the reliability of the program for a single cycle, R(c), is given by

$$R(c) = S(1n) * R(n).$$

Several important points should be made here. First, since the calculation of software reliability was computed based on operational path probabilities, the prediction made is limited to the scenario or mission described by those probabilities. Secondly, the value computed is cycle based. That is, it represents the probability of successfully completing the software one time. Many operational missions require the repeated cycling of a computer program throughout a given mission time. Equally important, software reliability must be expressed in a time-based reference if it is to be combined with hardware reliability to arrive at an overall mission reliability.

3.0 USE OF THE MARKOV PROCESS

The analyst must essentially accomplish the steps defined above. Having already determined the individual module reliabilities (Appendix E) and having already accomplished a path analysis (Appendix C), the matrices R and P can be constructed. The R matrix is simply a diagonal matrix containing the individual module reliabilities. That is R(nn) equals the reliability of the n'th module. All other entries in the R matrix are set to zero. The P matrix contains all the path probabilities. In the first row the analyst must list the probabilities of module 1 passing control to each of the modules (itself included). In the second row, the same is listed with respect to control leaving module 2.

Multiply R*P. Since R is a diagonal matrix, this step can be done by inspection simply by multiplying every entry of row i in P by R(ii).

The matrix product R*P is referred to as the Q matrix. The next step is to substract Q from the identity (diagonal matrix of 1's) and then compute the inverse of the I-Q matrix. Call it the S matrix. This is the only step which is computationaly difficult. Matrix inversion is difficult for all but the smallest problems. It is highly recommended that this step of the methodology be computerized.

The overall software reliability can be easily calculated from the S matrix. Since S(1n) is the probability of reaching module n, and since n is the last module, the product of S(1n) and R(n) is equal to the overall software reliability.

APPENDIX G

DETECTION AND WARNING SYSTEM

1.0 INTRODUCTION

To illustrate the application of the software portion of the combined Hardware/Software System Reliabiity Prediction Methodology, a relatively straightforward example is presented.

2.0 THE PROBLEM DESCRIPTION

The application is a simple search and warning system which continually scans an area of the atmosphere, evaluates returns, and issues a warning message when a detected object is found to be hostile. The hardware portion of the system consists of devices which can be directed to scan an area, perform an acquisition pattern or track a specific path as commanded by the computer. The computer software is required to analyze returns, issue move commands to the hardware and report hostile targets.

In the search mode the software is required to accept positional data from the hardware, correlate the individual returns and perform preliminary analysis of the potential target. If no target meets a preset correlation threshold or if none of the targets meet other threat conditions, the hardware is directed to remain in the search mode. If a potential target is detected, the software must initiate an acquisition mode.

In the acquisition mode, the software will be receiving returns from the hardware in accordance with a pre-defined pattern. It must identify those returns which correlate with the target being acquired and computer trajectory data for the target. When the system is able to predict the next position of the target within a given error tolerance, the software must issue a command to initiate track mode. If the system is unable to acquire the target within a prescribed time, acquistion is aborted and the operation is returned to the search mode.

In the track mode, the software issues pointing coordinates to the hardware, evaluates the returns, updates its trajectory estimates and evaluates the target's predicted launch and impact points. If the target is untrackable or if it is found to be non-hostile based on the trajectory analysis, the system is directed to return to the search mode. If it is found to be hostile and its predicted launch and impact points can be computed within a pre-determined confidence range, a message is sent, track is broken and the system is returned to the search mode.

The system is designed to be operational at least 95 percent of the time. That is, it may be shut down for 72 minutes every day for preventative maintenance. When the operator issues a shutdown command, the software will perform self-checking maintenance routines and issue the necessary commands to shut down the entire system. During operation, the software portion of the system must achieve a 90 percent reliability.

3.0 APPLYING THE METHODOLOGY

3.1 Preliminary Analysis

Based on the system requirements definition, a functional decomposition is accomplished, and it is determined that software design will consist of five functionally independent modules: search, acquire, track, report, and maintenance. The top level functional flow chart is shown in Figure G-1. As can be seen from the flow diagram, each module has a single entry point and one or more possible exits. The logical flow from one module to the next is determined by various decision points within each of the modules. The functional design of the modules as well as the entry and exit decisions are extracted directly from the performance requirements listed in the previous section.

Next, it is necessary to predict the individual reliability of each of the functional modules. A context flow analysis reveals inherent characteristics of the data and control flow within each module. Inherent processing analysis identifies those aspects of individual modules which can be expected to affect its reliability. Considerations of its size, application category, language level, etc. can be compared with historical data of similar applications. Likewise, the developmental characteristics affect the manner in which errors are either avoided or detected. The worksheets presented in Appendix I of this volume are used to predict individual module reliabilities. Figures G-2a through G-2d demonstrate the calculation of the reliability for the track module. A separate worksheet must be accomplished for each module in the system.

Next, a functional thread analysis is performed to determine the likelihood (probability) of each possible path being executed. This is accomplished by analysis of anticipated scenarios coupled with the functional design of the software. In essence, the analyst must predict how often each of the switching decisions will be made. For example, the requirements specified that the system would be shut down for 72 minutes per day for preventative maintenance. Therefore, the probability of going from the search mode to the self-check mode is equal to 0.05 (72 minutes in a 1440 minute day). Other probabilities, such as the percent of time that acquisition mode will be entered must be determined by the same analytical threat assessments that were used when it was determined that the system was needed in the first place.

Finally, the module-to-module path probabilities and the module reliabilities are entered on the functional flow diagram (Figure G-3) for later analysis. Note that in Figure G-3, the module names have been replaced by module numbers to facilitate the mathematics that are performed in the next section.

Armed with individual module reliabilities and path execution probabilities, it is now possible to compute the conditional probabilities that a module will operate successfully given that it was executed. It should be obvious that there is essentially an unlimited number of possible paths, and consequently, an unlimited number of calculations to be made. Fortunately, the scenario can be described by a Markov process which duplicates the transitions from one state to the next. Furthermore, once the mathematical transition matrix has been established, it can be manipulated to evaluate the infinite sum of conditional probabilities via a single matrix operation. The section that follows, utilizes the results of the path analysis and module reliability analysis to construct the Markov transition matrix and determine the reliability of the overall software component of the system.

3.2 Mathematical Computations

Based on the preliminary analyses of individual module reliabilities and the functional path analysis, we can summarize the problem using mathematical shorthand as follows:

$R_1 = 0.98$	Reliability of module 1
$R_2 = 0.97$	Reliability of module 2
$R_3 = 0.96$	Reliability of module 3
$R_4 = 0.95$	Reliability of module 4
$R_5 = 0.999$	Reliability of module 5 .

The path probabilities depicted on the flowchart in Figure G-3 can be represented as a matrix, P as follows:

$$P = \begin{bmatrix} 0.8555 & 0.095 & 0 & 0 & 0.05 \\ 0.45 & 0.45 & 0.1 & 0 & 0 \\ 0.45 & 0 & 0.45 & 0.1 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} .$$

The entry in the i'th row and j'th column represents the probability of module i passing control to module j.

It we were to attempt to evaluate all possible conditional probabilities, the number of calculations would be astronomical. The following is a very short list of some of the possible paths:

1-5	1-2-1-1-5	1-2-3-3-1-5
1-2-1-5	1-2-3-4-1-5	1-2-3-3-1-2-2-1-5
1-2-3-1-5	1-2-1-2-1-5	1-2-2-2-3-4-1-5.

Computation of even a single path requires the multiplication of all the path probabilities and module reliabilities involved in the path. For example, consider the path 1-2-3-4-1-5. The conditional probability, C, that it will be successfully accomplished is computed as follows:

$$\begin{aligned} C &= (R_1)(P_{12})(R_2)(P_{23})(R_3)(P_{34})(R_4)(P_{41})(R_1)(P_{15})(R_5) \\ &= (0.98)(0.095)(0.97)(0.1)(0.96)(0.1)(0.95)(1.0)(0.98)(0.05)(0.999) \\ &= 4.03\text{E-}5. \end{aligned}$$

At first glance it appears that this value is extremely small. However, this path represents only one of the essentially unlimited number of possible paths.

The reliability of the overall software subsystem is the sum of the conditional probabilities of all the possible paths. If the program is structured, it may be possible to compute the overall reliability by hand despite the fact that there may be an infinite number of paths. This would be accomplished by assuming that the effects of certain paths are negligible. Even so, the task is extremely tedious. If the program is not structured, the task of computing reliability by hand borders on the impossible.

The alternative is to use the Markov analysis in a manner described by Cheung [1]. Using the matrix P as defined above and the reliability matrix R defined as:

$$R = \begin{bmatrix} 0.98 & 0 & 0 & 0 & 0 \\ 0 & 0.97 & 0 & 0 & 0 \\ 0 & 0 & 0.96 & 0 & 0 \\ 0 & 0 & 0 & 0.96 & 0 \\ 0 & 0 & 0 & 0 & 0.999 \end{bmatrix} .$$

We compute the matrix $Q = R*P$. The matrix Q then is

$$Q = \begin{bmatrix} 0.838 & 0.093 & 0.000 & 0.000 & 0.049 \\ 0.437 & 0.437 & 0.099 & 0.000 & 0.000 \\ 0.433 & 0.000 & 0.432 & 0.096 & 0.000 \\ 0.950 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix} .$$

The matrix Q transforms the success/failure status of the software from one state to another. In order to determine overall software reliability, it is necessary to sum all possible transformations; i.e.,

$$S = I + Q^1 + Q^2 + Q^3 + Q^4 + \ldots\ldots\ldots + Q^i$$

where I is the initial state (the identity matrix) and i is unbounded. Let this sum be given by

$$S = \sum_{n=0}^{n=\infty} Q^n .$$

It can be shown that the sum of infinite terms can be evaluated as the inverse of the matrix (I - Q); i.e.,

$$S = (I - Q)^{-1}$$

For this particular example, the matrix is:

$$S = \begin{bmatrix} 13.294 & 2.196 & 0.375 & 0.036 & 0.651 \\ 12.406 & 3.824 & 0.653 & 0.063 & 0.608 \\ 12.246 & 2.023 & 2.106 & 0.202 & 0.600 \\ 12.630 & 2.087 & 0.356 & 1.034 & 0.619 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Evaluation of the matrix inversion described above can be very tedious when many modules are present. It is suggested that this step be accomplished on a computer. Figure G-4 illustrates the printout used to generate these sample results.

The reliability of the overall software subsystem can be easily computed as the probability of transferring control from the first node to the last node multiplied by the reliability of the last node. In this example:

$$\begin{aligned} R(s) &= S(1,5) * R5 \\ &= (0.651)(0.999) \\ &= 0.651. \end{aligned}$$

This example represents the probability that the software will perform all of its mission functions for a full day. It can then be combined with hardware reliability measurements to determine overall system reliability.

4.0 USING THE RESULTS

Having predicted the software mission reliability, an assessment of its acceptability must be made. If the prediction is unacceptable, a decision must be made as to where (in the system) improvements can be realized. In the case of software reliability improvement, the methodology can be utilized to test the effects of various design, development and testing philosophies by incorporating the pi factors associated with a proposed approach and reaccomplishing the mathematical computations. It can be used repeatedly to test tradeoffs before a commitment is made.

Additionally, the methodology can be used to periodically re-calculate reliability predictions and refine them as the development progresses and better estimates of module reliabilities and path probabilities are available.

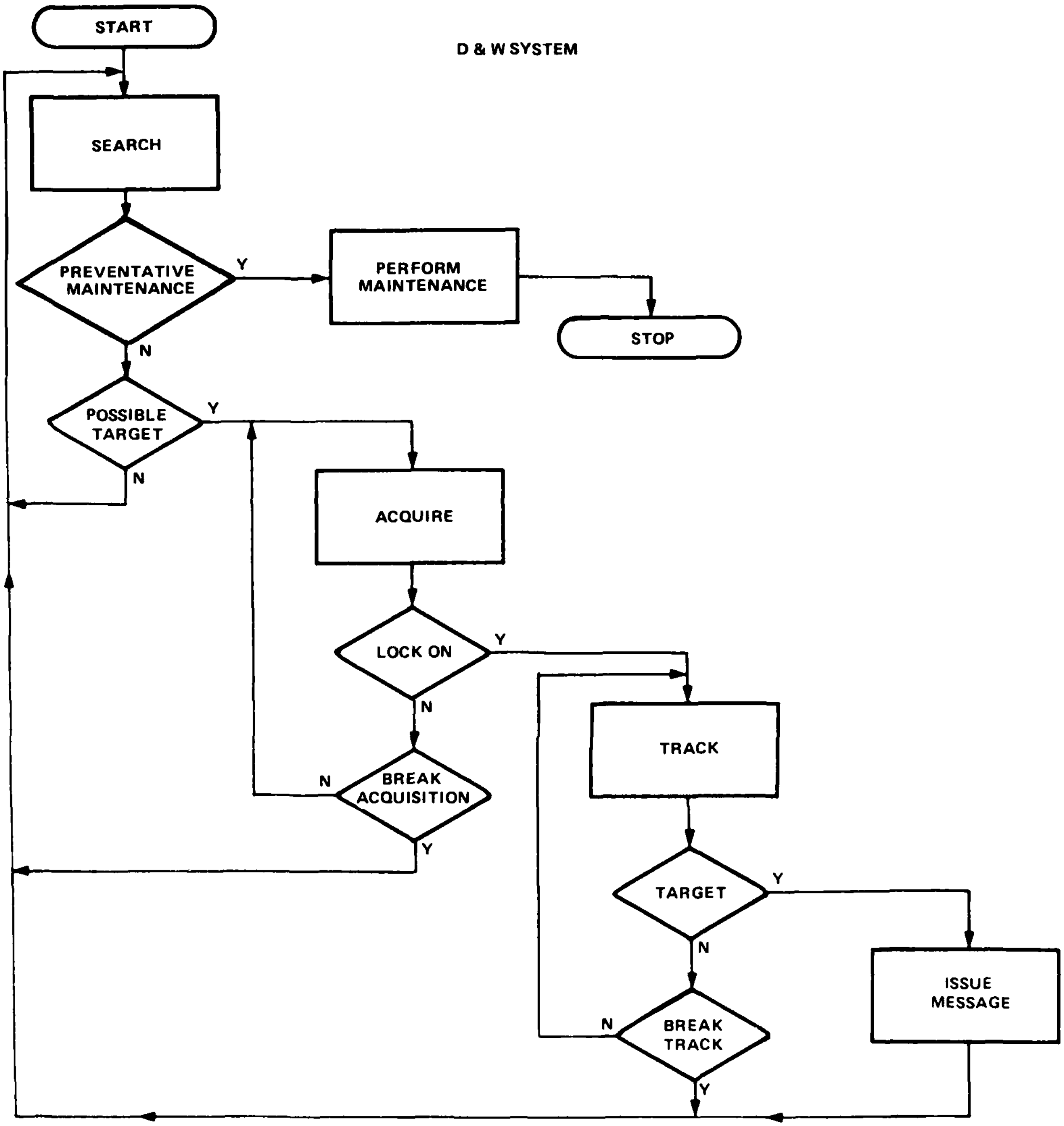


Figure G-1. <u>Functional Flow Chart</u>

WORKSHEET No. 1

INHERENT CHARACTERISTICS - C(1) NOMINAL CASE MODULE: 3

FACTOR	(check)	C(1)	C(2)	C(3)	C(4)
PREDOMINANTLY CONTROL		.372	.299	.192	.136
PREDOMINANTLY REAL TIME		.308	.308	.203	.166
PREDOMINANTLY INPUT/OUTPUT		.205	.256	.423	.105
PREDOMINANTLY INTERACTIVE		.270	.342	.264	.121
PREDOMINANTLY COMPUTATIONAL	✓	.277	.169	.133	.411
MANY DISTINCT OPERATIONAL MISSIONS		.385	.299	.175	.141
SEVERAL VARIATIONS OF OPERATIONAL MISSIONS		.362	.298	.178	.162
SINGLE OPERATIONAL MISSION	✓	.323	.251	.194	.212
MANY OPERATIONS REQUIRED - HIGHLY INTERRELATED		.385	.316	.151	.149
MANY OPERATIONS REQUIRED - RELATIVELY INDEPENDENT		.372	.236	.187	.191
FEW OPERATIONS REQUIRED - HIGHLY INTERRELATED		.346	.316	.168	.172
FEW OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	✓	.343	.240	.182	.216
EXTENSIVE HARDWARE INTERFACE REQUIREMENTS		.262	.425	.209	.104
MINIMAL HARDWARE INTERFACE REQUIREMENTS		.311	.284	.214	.175
EXTENSIVE SOFTWARE INTERFACE REQUIREMENTS		.284	.417	.175	.113
MINIMAL SOFTWARE INTERFACE REQUIREMENTS		.307	.275	.215	.189
EXTENSIVE HUMAN INTERFACE REQUIREMENTS		.270	.341	.267	.112
MINIMAL HUMAN INTERFACE REQUIREMENTS		.322	.269	.213	.178
WIDE RANGE OF ERROR-PRONE INPUTS		.303	.236	.283	.176
WIDE RANGE OF ERROR-FREE INPUTS		.304	.253	.247	.192
NARROW RANGE OF ERROR-PRONE INPUTS		.313	.236	.259	.180
NARROW RANGE OF ERROR-FREE INPUTS	✓	.311	.237	.248	.200
SUM OF THE VALUES CHECKED:		1.25	.90	.76	1.04
ABOVE SUM DIVIDED BY THE NUMBER OF CHECKS:		.314	.224	.189	.273
		C(1)	C(2)	C(3)	C(4)

Figure G-2a. Track Module Worksheet No. 1

WORKSHEET No. 2

ERROR AVOIDANCE TECHNIQUES - A(i) NOMINAL CASE MODULE: 3

FACTOR	(check)	1-A(1)	1-A(2)	1-A(3)	1-A(4)
INDEPENDENT QUALITY ASSURANCE ORGANIZATION		.646	.653	.692	.697
INDEPENDENT TEST ORGANIZATION	✓	.696	.662	.651	.646
INDEPENDENT VERIFICATION AND VALIDATION (IV&V)		.631	.661	.648	.622
USE OF A SOFTWARE SUPPORT LIBRARY		.779	.757	.772	.757
USE OF A SOFTWARE CONFIGURATION CONTROL BOARD		.775	.679	.777	.831
THOROUGH AND ENFORCED SOFTWARE DEVELOPMENT PLAN		.672	.652	.668	678
RIGIDLY CONTROLLED SYSTEM REQUIREMENTS SPEC		.646	.617	.677	.698
RIGIDLY CONTROLLED INTERFACE DESIGN SPEC	✓	.698	.466	.630	.769
RIGIDLY CONTROLLED SOFTWARE REQUIREMENTS SPEC	✓	.598	.605	.634	.660
RIGIDLY CONTROLLED SOFTWARE FUNCTIONAL DESIGN SPEC		.637	.612	.655	.656
RIGIDLY CONTROLLED SOFTWARE DETAILED DESIGN SPEC		.617	.608	.610	.597
REQUIREMENTS TRACEABILITY MATRIX		.645	.681	.738	.771
STRUCTURED ANALYSIS TOOLS		.663	.694	.717	.742
PROGRAM SPECIFICATION LANGUAGE (PSL)		.682	.710	.748	.752
PROGRAM DESIGN LANGUAGE (PDL)		.643	.687	.721	.714
HIGH ORDER LANGUAGE (HOL)		.674	.706	.698	.651
HIERARCHICAL, TOP-DOWN DESIGN		.617	.633	.685	.735
STRUCTURED DESIGN	✓	.619	.640	.677	.715
SINGLE FUNCTION MODULARIZATION		.625	.674	.698	.697
STRUCTURED CODE		.629	.695	.720	.663
USE OF AUTOMATIC MEASUREMENT TOOLS		.761	.780	.772	.755
USE OF AUTOMATIC TEST TOOLS		.699	.718	.713	.692
PRODUCT OF THE VALUES CHECKED:		.180	.119	.176	.234
ABOVE PRODUCT SUBTRACTED FROM ONE:		.820	.881	.824	.766
		A(1)	A(2)	A(3)	A(4)

Figure G-2b. Track Module Worksheet No. 2

WORKSHEET No. 3

ERROR DETECTION TECHNIQUES - D(i) NOMINAL CASE MODULE: 3

FACTOR	(check)	1-D(1)	1-D(2)	1-D(3)	1-D(4)
FREQUENT PEER WALKTHROUGHS		.548	.595	.621	.607
INFREQUENT PEER WALKTHROUGHS	✓	.749	.775	.805	.797
FREQUENT PROGRESS REVIEWS		.755	.780	.765	.802
INFREQUENT PROGRESS REVIEWS		.890	.890	.898	.912
FREQUENT QUALITY AUDITS		.712	.731	.733	.755
INFREQUENT QUALITY AUDITS		.862	.868	.880	.882
USE OF SOFTWARE PROBLEM REPORTS PRIOR TO PQT		.666	.668	.703	.665
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO PQT		.735	.722	.748	.742
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO FQT	✓	.755	.716	.763	.754
USE OF SPECIFICATION CHANGE NOTICES (SCN's)		.807	.779	.811	.829
USE OF ENGINEERING CHANGE NOTICES (ECN's)		.799	.769	.802	.818
SOFTWARE REQUIREMENTS REVIEW (SRR)		.686	.669	.720	.766
PRELIMINARY DESIGN REVIEW (PDR)	✓	.714	.679	.721	.773
CRITICAL DESIGN REVIEW (CDR)		.677	.672	.695	.738
TEST READINESS REVIEW (TRR)		.799	.766	.781	.797
FUNCTIONAL CONFIGURATION AUDIT (FCA)		.816	.782	.792	.817
PHYSICAL CONFIGURATION AUDIT (PCA)		.851	.826	.823	.855
INFORMAL UNIT-LEVEL TESTING	✓	.566	.682	.655	.544
PRELIMINARY QUALIFICATION TESTING (PQT)		.639	.659	.682	.679
FORMAL QUALIFICATION TESTING (FQT)	✓	.686	.700	.711	.698
SOFTWARE INTEGRATION TESTING		.651	.550	.616	.666
SYSTEM INTEGRATION TESTING		.670	.587	.633	.688
OPERATIONAL FIELD TESTING		.673	.648	.637	.686
PRODUCT OF THE VALUES CHECKED:		.157	.180	.206	.176
ABOVE PRODUCT SUBTRACTED FROM ONE:		.843	.820	.794	.824
		D(1)	D(2)	D(3)	D(4)

Figure G-2c. Track Module Worksheet No. 3

WORKSHEET No. 4

MODULE RELIABILITY CALCULATION - NOMINAL CASE MODULE: 3

$$ADC(i) = \frac{A(i)*C(i)}{1.00 - D(i)*[1.00 - A(i)]} = \frac{\text{substitute appropriate } A(i) \text{ and } C(i)}{\text{substitute appropriate } A(i) \text{ and } D(i)} = \text{calculate}$$

$$ADC(1) = \frac{A(1)*C(1)}{1.00 - D(1)*[1.00 - A(1)]} = \frac{.820\,(.314)}{1 - .843\,(1 - .820)} = .304$$

$$ADC(2) = \frac{A(2)*C(2)}{1.00 - D(2)*[1.00 - A(2)]} = \frac{.881\,(.224)}{1 - .820\,(1 - .881)} = .219$$

$$ADC(3) = \frac{A(3)*C(3)}{1.00 - D(3)*[1.00 - A(3)]} = \frac{.824\,(.189)}{1 - .794\,(1 - .824)} = .181$$

$$ADC(4) = \frac{A(4)*C(4)}{1.00 - D(4)*[1.00 - A(4)]} = \frac{.766\,(.273)}{1 - .824\,(1 - .766)} = .259$$

SUM ALL THE ADC(i) TERMS --> .963

(COPY)

CALCULATED ENHANCEMENT FACTOR - NOMINAL CASE .963

Figure G-2d. Track Module Worksheet No. 4

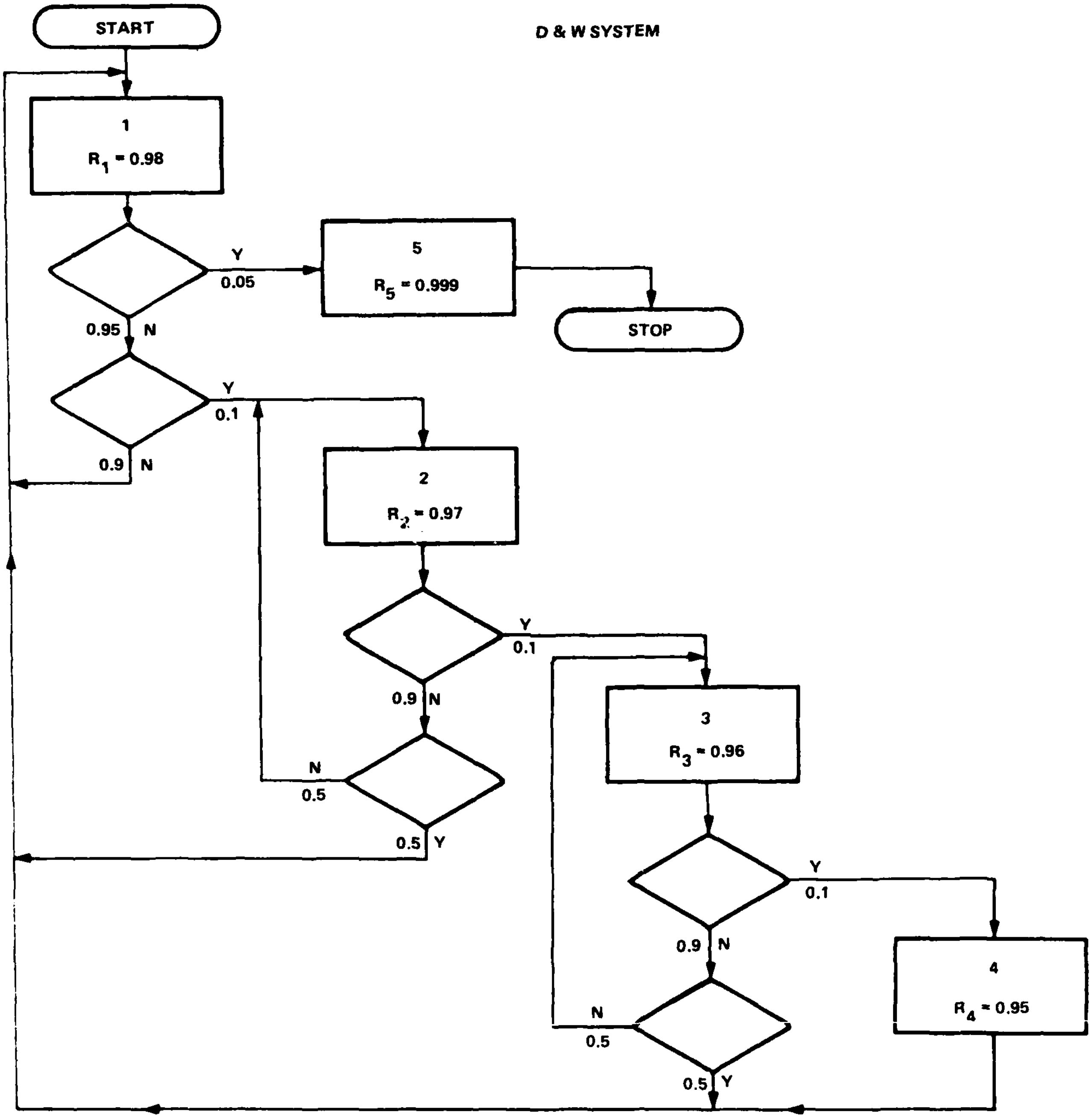


Figure G-3. Annotated Flow Chart

```
MODULE RELIABILITY MATRIX (R):

 0.980  0.970  0.960  0.950  0.999

PATH MATRIX (P):

 0.855  0.095  0.000  0.000  0.050
 0.450  0.450  0.100  0.000  0.000
 0.450  0.000  0.450  0.100  0.000
 1.000  0.000  0.000  0.000  0.000
 0.000  0.000  0.000  0.000  0.000

CONDITIONAL PATH MATRIX (G = RxP):

 0.838  0.093  0.000  0.000  0.049
 0.437  0.437  0.097  0.000  0.000
 0.432  0.000  0.432  0.096  0.000
 0.950  0.000  0.000  0.000  0.000
 0.000  0.000  0.000  0.000  0.000

IDENTITY - G MATRIX (W = I-G):

 0.162 -0.093  0.000  0.000 -0.049
-0.437  0.563 -0.097  0.000  0.000
-0.432  0.000  0.568 -0.096  0.000
-0.950  0.000  0.000  1.000  0.000
 0.000  0.000  0.000  0.000  1.000

SOLUTION MATRIX (S = INVERSE W):

13.294  2.196  0.375  0.036  0.651
12.406  3.824  0.653  0.063  0.608
12.246  2.023  2.106  0.203  0.600
12.630  2.087  0.356  1.034  0.619
 0.000  0.000  0.000  0.000  1.000

SOFTWARE RELIABILITY (R(n)*S(1,n)) = :   0.651
```

Figure G-4. Computer Output

APPENDIX H

ASSAULT BREAKER

1.0 INTRODUCTION

The Assault Breaker project was chosen to illustrate a typical application of the reliability methodology to a real-time system. Assault Breaker is a recently completed Martin Marietta program which involved a tactical missile and its controlling software.

2.0 THE PROBLEM DESCRIPTION

The Assault Breaker software was tasked with launching the missile, steering it to a target area and dispensing submunitions. The hardware portion of the system interfaced with the software by providing positional data and by accepting guidance commands.

The flight software was decomposed into nine distinct functions or modules each requiring execution periodically as shown below:

1.	Program initialization	1 time
2.	Executive control program	200 Hz
3.	Digital autopilot	100 Hz
4.	Flight control	10 Hz
5.	Navigational filter	10 Hz
6.	Antenna select	10 Hz
7.	Target processor	10 Hz
8.	Dispense	10 Hz
9.	Steering (Guidance)	10 Hz

a. Launch initialization
b. Initial turn steering
c. Midcourse steering
d. Terminal steering.

Since the software was required to operate in real-time, an extensive analysis was performed to determine its best overall structure. A brief recap of that analysis follows.

Analysis of the missile rigid and dynamic body bending modes dictated a digital autopilot operating at a 200-Hz rate to maintain missile stability. Further analysis of the dispense accuracy requirements resulted in dispense commands being initiated at a 200-Hz rate and being tuned accurately from time of launch including missile variations from the nominal flight trajectory. Maximum flight from launch is 180 seconds.

Launch control and self-test, launch initialization, and program initialization modules are all completed prior to launch when none of the missile flight programs are operating. These modules have lower priority for reliability as there are many checks built into the system operation to verify the accuracy of the missile prior to launch.

Performing a timing and loading analysis of the program functions and applying these timing results to the reliability model resulted in a very poor reliability value for the overall program. The accuracy requirements with the supersonic stability requirements dictated an accurate time line control of function completion to eliminate phase delay errors in the uncertainty of the time that any particular computation was completed.

The results of this analysis led to the selection of a structured executive format rather than a task driven executive. Using this structured technique with the timing analysis, the functions were decomposed into modules to be performed in a time controlled sequence. The autopilot function was a known function in this structure which was required to be performed every five milliseconds (200 Hz). Worst case analysis of the timing of this module was shown to be 700 microseconds. The executive module was designed not to exceed one millisecond and actually used 720 microseconds worst case.

Each of the functions was decomposed into submodules with a design goal of one millisecond and not to exceed 1.5 milliseconds. The result was that of each five millisecond period, the worst case time usable required to complete all modules was less than 3.5 milliseconds. This resulted in a probability of one that each module would be completed in its assigned time slot, providing no hardware errors occured. The executive program was designed to verify that no real-time module was in operation when the five millisecond real-time clock interrupted.

By structuring the minor cycles of five milliseconds into 20 cycles to a major cycle (100 milliseconds) the accuracy constraints of the missile requirements were met.

3.0 APPLYING THE METHODOLOGY

3.1 Preliminary Analysis

Figure H-1 depicts the final functional flow diagram of the flight software. It is representative of many real-time application programs which have been constructed in a top-down manner.

Figures H-2a through H-2d depict the worksheets used to determine the reliability of the Antenna Select module. It should be noted that on the Assault Breaker project, many reliability enhancing techniques were employed. This resulted in relatively high individual reliabilities. Each of the modules was evaluated with the following results:

No.	Module Name	Reliability
1.	Program initialization	0.999
2.	Executive control program	0.996
3.	Digital autopilot	0.990
4.	Flight control	0.987
5.	Navigational filter	0.998
6.	Antenna select	0.982
7.	Target processor	0.992
8.	Dispense	0.999
9.	Steering (guidance)	0.992.

Functional thread analysis was relatively easy. As was described above and depicted in Figure H-1, the system is initialized one time; and subsequently, the Executive module directs the logical flow to the Autopilot and then to one of the six functional modules. Each of those modules returns control to the Executive. Investigation into the timing analysis that had been performed revealed that several of the modules required more than one cycle to complete. Specifically, each module was allocated a specific number of cycles for completion of its functions. The path probabilities used for the reliability analysis were therefore computed as the proportion of a 20-cycle period that was allocated to each module:

No.	Module Name	No. Cycles Per Period	Path Probability From Autopilot
4.	Flight control	7	7/20 = 0.35
5.	Navigational filter	3	3/20 = 0.15
6.	Antenna select	1	1/20 = 0.05
7.	Target processor	3	3/20 = 0.15
8.	Dispense	1	1/20 = 0.05
9.	Steering (guidance)	5	5/20 = 0.25.

The only other path probabilities to be considered involved determining how to stop the program. Obviously, with a missile system, the computer software will continue processing until the missile is destroyed. To alleviate the possibility of endless looping (in the methodology), a fictitious module was created. It was labeled as the Termination module on Figure H-1 and was arbitrarily assigned a reliability of 0.999 to minimize its impact on the analysis. Since it is only executed one time, its effect to the analysis is negligible.

The next major consideration to be made was to probabilistically define the path which leads to termination. Review of the functional decomposition revealed that the software performed under four specific modes of operation. Since these mode changes are the points where relatively drastic changes in the input domain are seen by the software, it was decided that the path probabilities from the Executive should be computed based on the five (including Termination) mode changes accomplished during the intended mission. That is, the value of the path probability going from the Executive to Termination was set to 0.20 (one mode/five possible) and the path to the Autopilot was set to 0.80 (four modes/five possible).

Finally, the module-to-module path probabilities and the module reliabilities were entered on the functional flow diagram (Figure H-3) for later analysis. Note that in Figure H-3, the module names have been replaced by module numbers to facilitate the mathematics that are performed in the next section.

Armed with individual module reliabilities and path execution probabilities, the reliability prediction methodology was applied exactly as already explained in Appendix G. The output of the analysis is shown in Figure H-4. The predicted reliability for the Assault Breaker Flight Software was calculated to be 0.911 for the mission scenario described above.

Although there has been insufficient test points to establish the validity of the prediction and hence the validity of the methodology, the little data that is available is quite interesting. Assault Breaker flew 10 flights during demonstration testing. There was only one failure and the computer software was not charged. However, the correction to the problem was accommodated by a software enhancement to perform additional checks prior to launch. If we surmise that the enhancement should have been in the original software design, we would be forced to charge the software for the failure, yielding a software mission reliability value of 0.90, almost exactly what the methodology predicted.

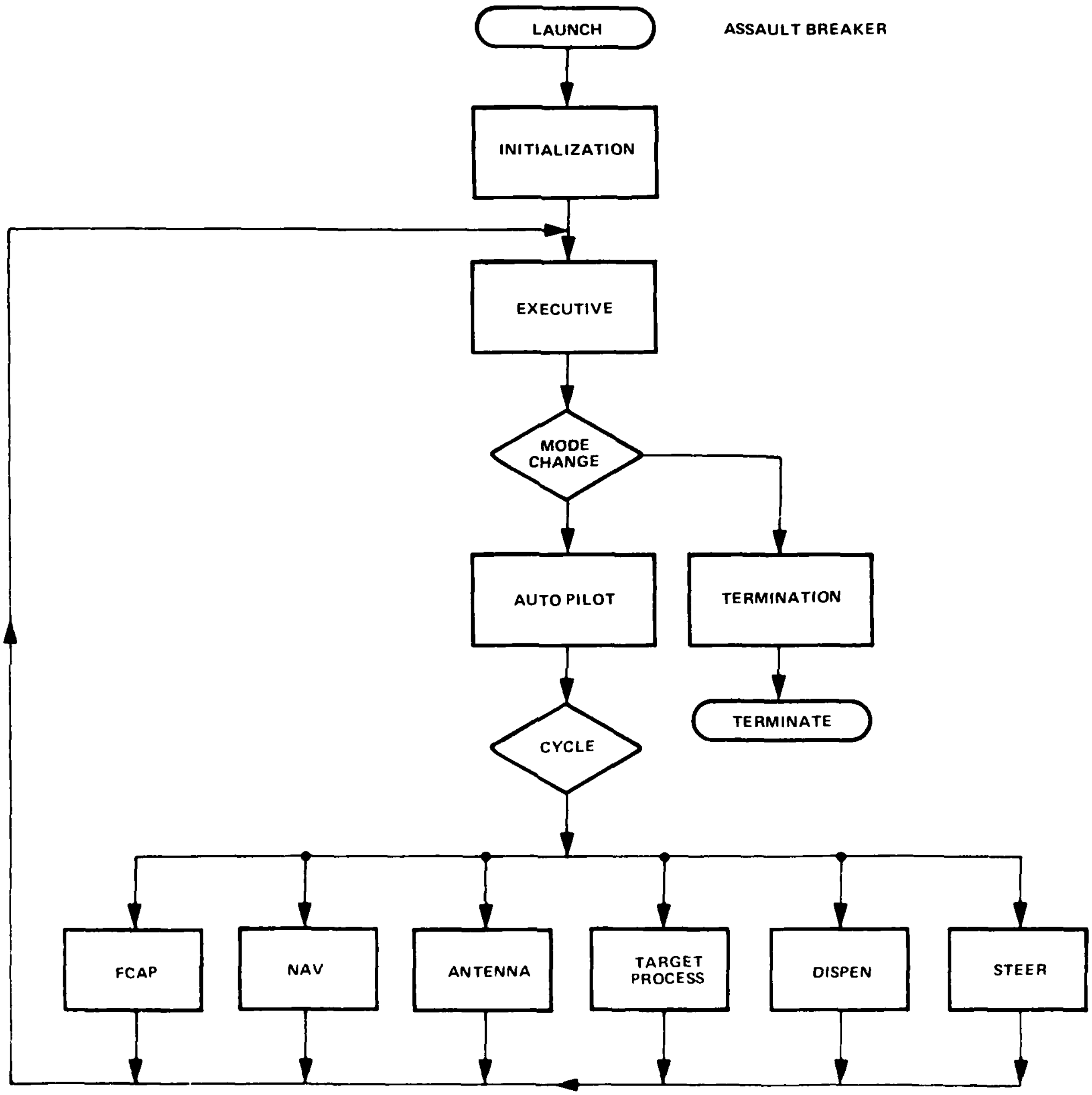


Figure H-1. Functional Flow Diagram

WORKSHEET No. 1

INHERENT CHARACTERISTICS - C(i) NOMINAL CASE MODULE: #6 ANTENNA SELECT

FACTOR	(check)	C(1)	C(2)	C(3)	C(4)
PREDOMINANTLY CONTROL		.372	.299	.192	.136
PREDOMINANTLY REAL TIME		.308	.308	.203	.166
PREDOMINANTLY INPUT/OUTPUT		.205	.256	.423	.105
PREDOMINANTLY INTERACTIVE		.270	.342	.264	.121
PREDOMINANTLY COMPUTATIONAL	✓	.277	.169	.133	.411
MANY DISTINCT OPERATIONAL MISSIONS		.385	.299	.175	.141
SEVERAL VARIATIONS OF OPERATIONAL MISSIONS		.362	.298	.178	.162
SINGLE OPERATIONAL MISSION	✓	.323	.251	.194	.212
MANY OPERATIONS REQUIRED - HIGHLY INTERRELATED		.385	.316	.151	.149
MANY OPERATIONS REQUIRED - RELATIVELY INDEPENDENT		.372	.236	.187	.191
FEW OPERATIONS REQUIRED - HIGHLY INTERRELATED		.346	.316	.168	.172
FEW OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	✓	.343	.240	.182	.218
EXTENSIVE HARDWARE INTERFACE REQUIREMENTS		.262	.425	.209	.104
MINIMAL HARDWARE INTERFACE REQUIREMENTS	✓	.311	.284	.214	.175
EXTENSIVE SOFTWARE INTERFACE REQUIREMENTS		.284	.417	.175	.113
MINIMAL SOFTWARE INTERFACE REQUIREMENTS	✓	.307	.275	.215	.169
EXTENSIVE HUMAN INTERFACE REQUIREMENTS		.270	.341	.267	.112
MINIMAL HUMAN INTERFACE REQUIREMENTS		.322	.269	.213	.178
WIDE RANGE OF ERROR-PRONE INPUTS		.303	.236	.283	.176
WIDE RANGE OF ERROR-FREE INPUTS		.304	.253	.247	.192
NARROW RANGE OF ERROR-PRONE INPUTS		.313	.236	.259	.180
NARROW RANGE OF ERROR-FREE INPUTS		.311	.237	.248	.200
SUM OF THE VALUES CHECKED:		1.561	1.219	.938	1.205
ABOVE SUM DIVIDED BY THE NUMBER OF CHECKS:		.312	.244	.188	.241
		C(1)	C(2)	C(3)	C(4)

Figure H-2a. Antenna Select Module Worksheet No. 1

WORKSHEET No. 2

ERROR AVOIDANCE TECHNIQUES - A(1) NOMINAL CASE MODULE: #6 ANTENNA SELECT

FACTOR	(check)	1-A(1)	1-A(2)	1-A(3)	1-A(4)
INDEPENDENT QUALITY ASSURANCE ORGANIZATION		.646	.653	.692	.697
INDEPENDENT TEST ORGANIZATION		.696	.662	.651	.646
INDEPENDENT VERIFICATION AND VALIDATION (IV&V)		.631	.661	.648	.622
USE OF A SOFTWARE SUPPORT LIBRARY	✓	.779	.757	.772	.757
USE OF A SOFTWARE CONFIGURATION CONTROL BOARD	✓	.775	.679	.777	.831
THOROUGH AND ENFORCED SOFTWARE DEVELOPMENT PLAN		.672	.652	.668	.678
RIGIDLY CONTROLLED SYSTEM REQUIREMENTS SPEC		.646	.617	.677	.698
RIGIDLY CONTROLLED INTERFACE DESIGN SPEC		.698	.466	.630	.769
RIGIDLY CONTROLLED SOFTWARE REQUIREMENTS SPEC	✓	.598	.605	.634	.660
RIGIDLY CONTROLLED SOFTWARE FUNCTIONAL DESIGN SPEC	✓	.637	.612	.655	.656
RIGIDLY CONTROLLED SOFTWARE DETAILED DESIGN SPEC		.617	.608	.610	.597
REQUIREMENTS TRACEABILITY MATRIX	✓	.645	.681	.738	.771
STRUCTURED ANALYSIS TOOLS		.663	.694	.717	.742
PROGRAM SPECIFICATION LANGUAGE (PSL)		.682	.710	.748	.752
PROGRAM DESIGN LANGUAGE (PDL)		.643	.687	.721	.714
HIGH ORDER LANGUAGE (HOL)		.674	.706	.698	.651
HIERARCHICAL, TOP-DOWN DESIGN	✓	.617	.633	.665	.735
STRUCTURED DESIGN	✓	.619	.640	.677	.715
SINGLE FUNCTION MODULARIZATION	✓	.625	.674	.698	.697
STRUCTURED CODE		.629	.695	.720	.663
USE OF AUTOMATIC MEASUREMENT TOOLS		.761	.780	.772	.755
USE OF AUTOMATIC TEST TOOLS		.699	.718	.713	.692

	A(1)	A(2)	A(3)	A(4)
PRODUCT OF THE VALUES CHECKED:	.035	.035	.059	.076
ABOVE PRODUCT SUBTRACTED FROM ONE:	.965	.965	.941	.924

Figure H-2b. Antenna Select Module Worksheet No. 2

WORKSHEET No. 3

ERROR DETECTION TECHNIQUES - D(i) NOMINAL CASE

MODULE: #6 ANTENNA SELECT

FACTOR	(check)	1-D(1)	1-D(2)	1-D(3)	1-D(4)
FREQUENT PEER WALKTHROUGHS	✓	.548	.595	.621	.607
INFREQUENT PEER WALKTHROUGHS		.749	.775	.805	.797
FREQUENT PROGRESS REVIEWS	✓	.755	.780	.785	.802
INFREQUENT PROGRESS REVIEWS		.890	.890	.898	.912
FREQUENT QUALITY AUDITS		.712	.731	.733	.755
INFREQUENT QUALITY AUDITS		.862	.868	.860	.867
USE OF SOFTWARE PROBLEM REPORTS PRIOR TO PQT	✓	.666	.668	.703	.685
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO PQT		.735	.722	.748	.742
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO FQT		.755	.716	.763	.754
USE OF SPECIFICATION CHANGE NOTICES (SCN's)		.807	.779	.811	.829
USE OF ENGINEERING CHANGE NOTICES (ECN's)		.799	.769	.802	.818
SOFTWARE REQUIREMENTS REVIEW (SRR)	✓	.686	.669	.720	.766
PRELIMINARY DESIGN REVIEW (PDR)	✓	.714	.679	.721	.773
CRITICAL DESIGN REVIEW (CDR)	✓	.677	.672	.695	.738
TEST READINESS REVIEW (TRR)	✓	.799	.766	.781	.791
FUNCTIONAL CONFIGURATION AUDIT (FCA)		.818	.782	.792	.817
PHYSICAL CONFIGURATION AUDIT (PCA)		.851	.826	.823	.855
INFORMAL UNIT-LEVEL TESTING	✓	.566	.682	.655	.544
PRELIMINARY QUALIFICATION TESTING (PQT)		.639	.659	.682	.679
FORMAL QUALIFICATION TESTING (FQT)		.686	.700	.711	.698
SOFTWARE INTEGRATION TESTING		.651	.550	.618	.666
SYSTEM INTEGRATION TESTING		.670	.587	.633	.688
OPERATIONAL FIELD TESTING		.673	.648	.637	.668
PRODUCT OF THE VALUES CHECKED:		.041	.049	.063	.063
ABOVE PRODUCT SUBTRACTED FROM ONE:		.959	.951	.937	.937
		D(1)	D(2)	D(3)	D(4)

Figure H-2c. Antenna Select Module Worksheet No. 3

WORKSHEET No. 4

MODULE RELIABILITY CALCULATION - NOMINAL CASE MODULE: #6 ANTENNA SELECT

$$ADC(i) = \frac{A(i)*C(i)}{1.00 - D(i)*[1.00 - A(i)]} = \frac{\text{substitute appropriate } A(i) \text{ and } C(i)}{\text{substitute appropriate } A(i) \text{ and } D(i)} = \text{calculate}$$

$$ADC(1) = \frac{A(1)*C(1)}{1.00 - D(1)*[1.00 - A(1)]} = \frac{.965(.312)}{1 - .959(1 - .965)} = .311$$

$$ADC(2) = \frac{A(2)*C(2)}{1.00 - D(2)*[1.00 - A(2)]} = \frac{.965(.244)}{1 - .951(1 - .965)} = .244$$

$$ADC(3) = \frac{A(3)*C(3)}{1.00 - D(3)*[1.00 - A(3)]} = \frac{.941(.188)}{1 - .937(1 - .941)} = .187$$

$$ADC(4) = \frac{A(4)*C(4)}{1.00 - D(4)*[1.00 - A(4)]} = \frac{.924(.241)}{1 - .937(1 - .924)} = .240$$

SUM ALL THE ADC(i) TERMS --> .982

Figure H-2d. Antenna Select Module Worksheet No. 4

(COPY)

CALCULATED ENHANCEMENT FACTOR - NOMINAL CASE .982

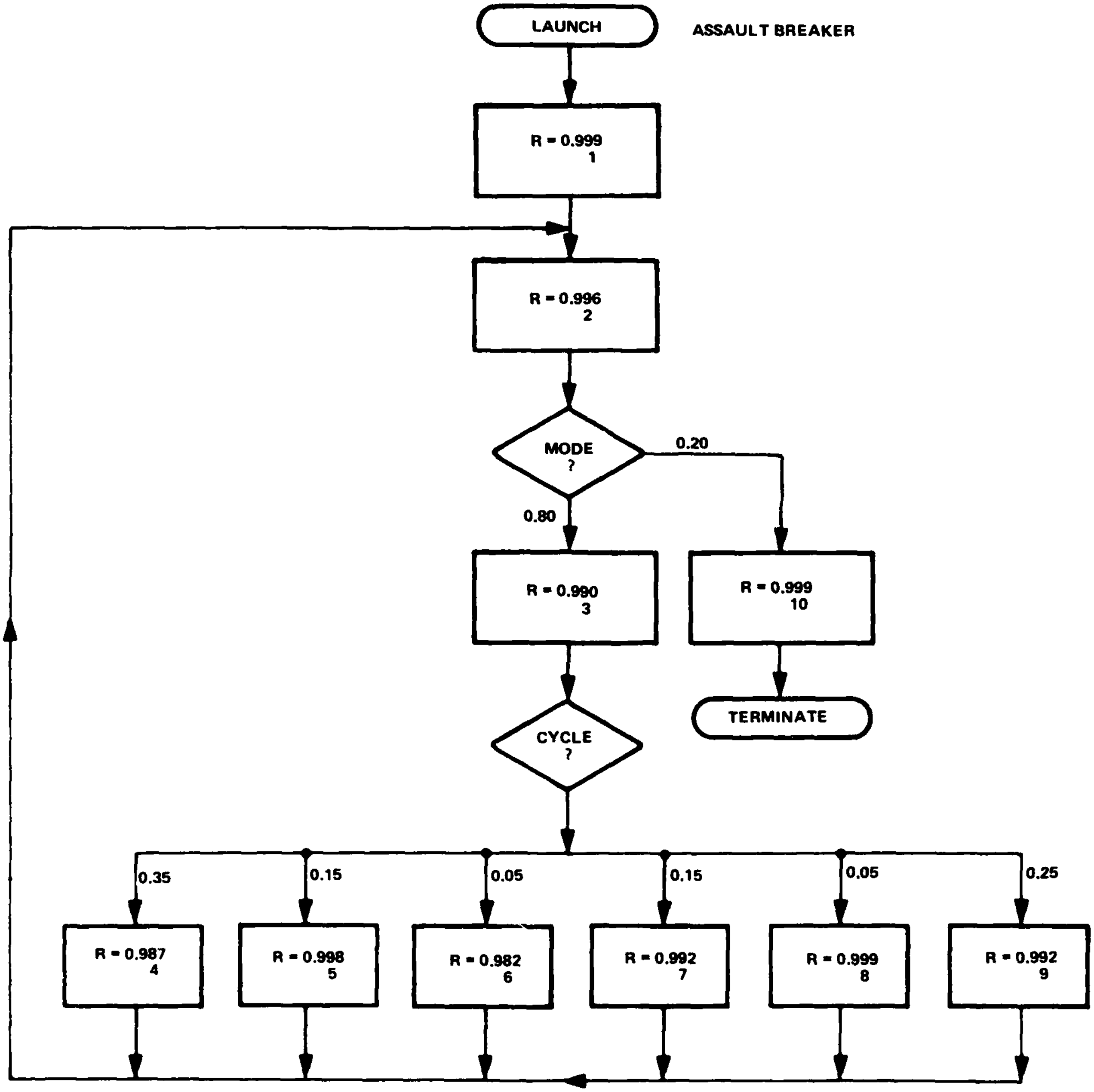


Figure H-3. Annotated Flow Chart

MODULE RELIABILITY MATRIX (R):

0.999	0.996	0.990	0.987	0.998	0.982	0.992	0.999	0.992	0.999

PATH MATRIX (P):

0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.800	0.000	0.000	0.000	0.000	0.000	0.000	0.200
0.000	0.000	0.000	0.350	0.150	0.050	0.150	0.050	0.250	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

CONDITIONAL PATH MATRIX (G = RxP):

0.000	0.999	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.797	0.000	0.000	0.000	0.000	0.000	0.000	0.199
0.000	0.000	0.000	0.347	0.149	0.049	0.149	0.049	0.248	0.000
0.000	0.987	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.998	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.982	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.992	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.999	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.992	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IDENTITY - G MATRIX (W = I-G):

1.000	-0.999	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	-0.797	0.000	0.000	0.000	0.000	0.000	0.000	-0.199
0.000	0.000	1.000	-0.347	-0.149	-0.049	-0.149	-0.049	-0.248	0.000
0.000	-0.987	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	-0.998	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
0.000	-0.982	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
0.000	-0.992	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
0.000	-0.999	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
0.000	-0.992	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

SOLUTION MATRIX (S = INVERSE W):

1.000	4.577	3.647	1.264	0.542	0.181	0.542	0.181	0.903	0.912
0.000	4.582	3.651	1.265	0.542	0.181	0.542	0.181	0.904	0.913
0.000	4.495	4.582	1.588	0.680	0.227	0.680	0.227	1.134	0.895
0.000	4.522	3.603	1.248	0.535	0.178	0.535	0.178	0.892	0.901
0.000	4.572	3.643	1.262	1.541	0.180	0.541	0.180	0.902	0.911
0.000	4.499	3.585	1.242	0.532	1.177	0.532	0.177	0.887	0.896
0.000	4.545	3.621	1.255	0.538	0.179	1.538	0.179	0.896	0.905
0.000	4.577	3.647	1.264	0.542	0.181	0.542	1.181	0.903	0.912
0.000	4.545	3.621	1.255	0.538	0.179	0.538	0.179	1.896	0.905
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

SOFTWARE RELIABILITY (R(n)*S(1,n)) = : 0.911

Figure H-4. Computer Output

APPENDIX I

WORKSHEETS

1.0 EXPLANATION OF WORKSHEET TERMS

1.1 Error Categories

Each of the worksheets contains four sets of calculations, identified via the subscripts 1 through 4. These subscripts are used to separate the effects of four different error types as follows:

1 LOGICAL ERRORS - This category includes all instances where a particular function is incorrect, inadequate or missing due to insufficient requirements definition, design errors or omissions, or implementation errors.

2 INTERFACE ERRORS - This category includes all instances where a required function is not implemented properly due to improper communication between system components. All possible interfaces are included in this category:

- Software/Software: Includes errors which occur between software components of the system such as when one program unit fails to call, calls in the wrong sequence, or otherwise improperly calls another program unit. Also included are all errors resulting from the improper sharing or passing of data and/or control variables between program units.

- Software/Hardware: Includes all errors which result in loss of data or untimely exchange of data between system hardware and embedded software. Included are situations where buffers become saturated or computation cycles exceed their timing allocations. Also included are errors caused by improper data exchange between system hardware and embedded software.

- Software/Human: See Input/Output Errors.

3 INPUT/OUTPUT ERRORS - This category includes all instances where a required function is not properly accomplished due to the manner in which input or output is implemented. For purposes of this survey, include in this category all software/human interfaces. For example, on input, the software may either accept improper commands or reject proper ones. On output, the software may generate erroneous or ambiguous messages to the operator.

4 COMPUTATIONAL ERRORS - These are calculation errors, which include: errors of omission such as uninitialized variables; mathematical errors such as incorrect expressions, conversion and truncation; and programming errors such as improper use of indices, variables and overlays.

1.2 Worst, Nominal and Best Cases

Included in this appendix are three sets of worksheets (four sheets each). Each set includes numerical values for inherent characteristics, avoidance techniques and detection techniques as well as a final calculation sheet. The sets are labeled worst, nominal and best case, respectively. The values were calculated statistically from the survey data described in detail in Volume I of this report. The nominal case represents the mean response to the survey. Best and worst are the 1-sigma values from the same survey.

1.3 Worksheets

The instructions for the worksheets are self-evident. The analyst need only check the appropriate characteristic or technique and perform the indicated operations. All four worksheets are based on the equations described in Appendix E of this volume and derived in Volume I.

On Worksheet 1, values for C(i) are listed. They represent the error type distribution expected due to each listed factor.

On Worksheet 2, the values listed are not the values of A(i) but rather 1.0 - A(i). Reference to Appendix D will show that the calculation of the overall avoidance effectiveness requires computing the product of the Non-avoidance probabilities of each technique used. The representation used in the worksheet was chosen to simplify this calculation. When the worksheet is completed in accordance with the instruction listed, the overall avoidance factors will result.

By the same rationale as above, Worksheet 3 lists the values on Non-detection probabilities. When the worksheet is completed in accordance with the instruction listed, the overall detection factors will result.

Worksheet 4 provides a straightforward implementation of the enhancement factor as defined in Appendix D.

WORKSHEET No. 1

INHERENT CHARACTERISTICS - C(i) WORST CASE MODULE: --

FACTOR	(check)	C(1)	C(2)	C(3)	C(4)
PREDOMINANTLY CONTROL	-------	.372	.299	.192	.138
PREDOMINANTLY REAL TIME	-------	.308	.308	.203	.166
PREDOMINANTLY INPUT/OUTPUT	-------	.205	.256	.423	.105
PREDOMINANTLY INTERACTIVE	-------	.270	.342	.264	.121
PREDOMINANTLY COMPUTATIONAL	-------	.277	.169	.133	.411
MANY DISTINCT OPERATIONAL MISSIONS	-------	.385	.299	.175	.141
SEVERAL VARIATIONS OF OPERATIONAL MISSIONS	-------	.362	.298	.178	.162
SINGLE OPERATIONAL MISSION	-------	.323	.251	.194	.212
MANY OPERATIONS REQUIRED - HIGHLY INTERRELATED	-------	.385	.316	.151	.149
MANY OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	-------	.372	.236	.187	.191
FEW OPERATIONS REQUIRED - HIGHLY INTERRELATED	-------	.346	.316	.168	.172
FEW OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	-------	.343	.240	.182	.218
EXTENSIVE HARDWARE INTERFACE REQUIREMENTS	-------	.262	.425	.209	.104
MINIMAL HARDWARE INTERFACE REQUIREMENTS	-------	.311	.284	.214	.175
EXTENSIVE SOFTWARE INTERFACE REQUIREMENTS	-------	.284	.417	.175	.113
MINIMAL SOFTWARE INTERFACE REQUIREMENTS	-------	.307	.275	.215	.189
EXTENSIVE HUMAN INTERFACE REQUIREMENTS	-------	.270	.341	.267	.112
MINIMAL HUMAN INTERFACE REQUIREMENTS	-------	.322	.269	.213	.178
WIDE RANGE OF ERROR-PRONE INPUTS	-------	.303	.236	.283	.178
WIDE RANGE OF ERROR-FREE INPUTS	-------	.304	.253	.247	.192
NARROW RANGE OF ERROR-PRONE INPUTS	-------	.313	.236	.259	.180
NARROW RANGE OF ERROR-FREE INPUTS	-------	.311	.237	.248	.200
SUM OF THE VALUES CHECKED:		----------	----------	----------	----------
ABOVE SUM DIVIDED BY THE NUMBER OF CHECKS:		----------	----------	----------	----------
		C(1)	C(2)	C(3)	C(4)

WORKSHEET No. 2

ERROR AVOIDANCE TECHNIQUES - A(i) WORST CASE MODULE: --

FACTOR	(check)	1- A(1)	1-A(2)	1-A(3)	1-A(4)
INDEPENDENT QUALITY ASSURANCE ORGANIZATION	-------	.704	.706	.747	.755
INDEPENDENT TEST ORGANIZATION	-------	.749	.715	.705	.699
INDEPENDENT VERIFICATION AND VALIDATION (IV&V)	-------	.685	.712	.704	.678
USE OF A SOFTWARE SUPPORT LIBRARY	-------	.821	.800	.814	.803
USE OF A SOFTWARE CONFIGURATION CONTROL BOARD	-------	.819	.729	.820	.870
THOROUGH AND ENFORCED SOFTWARE DEVELOPMENT PLAN	-------	.723	.702	.718	.731
RIGIDLY CONTROLLED SYSTEM REQUIREMENTS SPEC	-------	.695	.666	.724	.747
RIGIDLY CONTROLLED INTERFACE DESIGN SPEC	-------	.747	.514	.680	.817
RIGIDLY CONTROLLED SOFTWARE REQUIREMENTS SPEC	-------	.647	.653	.682	.705
RIGIDLY CONTROLLED SOFTWARE FUNCTIONAL DESIGN SPEC	-------	.683	.660	.703	.703
RIGIDLY CONTROLLED SOFTWARE DETAILED DESIGN SPEC	-------	.665	.659	.661	.647
REQUIREMENTS TRACEABILITY MATRIX	-------	.702	.728	.786	.819
STRUCTURED ANALYSIS TOOLS	-------	.714	.740	.766	.788
PROGRAM SPECIFICATION LANGUAGE (PSL)	-------	.731	.758	.792	.801
PROGRAM DESIGN LANGUAGE (PDL)	-------	.690	.733	.767	.761
HIGH ORDER LANGUAGE (HOL)	-------	.721	.752	.747	.698
HIERARCHICAL, TOP-DOWN DESIGN	-------	.664	.678	.732	.781
STRUCTURED DESIGN	-------	.665	.688	.726	.762
SINGLE FUNCTION MODULARIZATION	-------	.675	.726	.750	.742
STRUCTURED CODE	-------	.677	.740	.769	.714
USE OF AUTOMATIC MEASUREMENT TOOLS	-------	.806	.824	.816	.805
USE OF AUTOMATIC TEST TOOLS	-------	.749	.766	.760	.743

PRODUCT OF THE VALUES CHECKED:	----------	----------	----------	----------
ABOVE PRODUCT SUBTRACTED FROM ONE:	---------- A(1)	---------- A(2)	---------- A(3)	---------- A(4)

WORKSHEET No. 3

ERROR DETECTION TECHNIQUES - D(i) WORST CASE MODULE: ____________________

FACTOR	(check)	1-D(1)	1-D(2)	1-D(3)	1-D(4)
FREQUENT PEER WALKTHROUGHS	-------	.595	.645	.672	.657
INFREQUENT PEER WALKTHROUGHS	-------	.785	.813	.840	.832
FREQUENT PROGRESS REVIEWS	-------	.802	.823	.830	.844
INFREQUENT PROGRESS REVIEWS	-------	.912	.914	.922	.933
FREQUENT QUALITY AUDITS	-------	.757	.778	.779	.799
INFREQUENT QUALITY AUDITS	-------	.891	.896	.905	.907
USE OF SOFTWARE PROBLEM REPORTS PRIOR TO PQT	-------	.719	.719	.755	.740
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO PQT	-------	.779	.768	.792	.788
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO FQT	-------	.801	.770	.810	.802
USE OF SPECIFICATION CHANGE NOTICES (SCN's)	-------	.849	.822	.850	.869
USE OF ENGINEERING CHANGE NOTICES (ECN's)	-------	.843	.816	.846	.861
SOFTWARE REQUIREMENTS REVIEW (SRR)	-------	.735	.714	.766	.812
PRELIMINARY DESIGN REVIEW (PDR)	-------	.755	.720	.762	.815
CRITICAL DESIGN REVIEW (CDR)	-------	.719	.713	.738	.780
TEST READINESS REVIEW (TRR)	-------	.837	.807	.821	.838
FUNCTIONAL CONFIGURATION AUDIT (FCA)	-------	.860	.825	.836	.859
PHYSICAL CONFIGURATION AUDIT (PCA)	-------	.888	.862	.859	.891
INFORMAL UNIT-LEVEL TESTING	-------	.614	.734	.707	.593
PRELIMINARY QUALIFICATION TESTING (PQT)	-------	.686	.703	.727	.728
FORMAL QUALIFICATION TESTING (FQT)	-------	.733	.744	.758	.749
SOFTWARE INTEGRATION TESTING	-------	.696	.595	.663	.716
SYSTEM INTEGRATION TESTING	-------	.718	.632	.680	.740
OPERATIONAL FIELD TESTING	-------	.706	.680	.669	.722
PRODUCT OF THE VALUES CHECKED:		----------	----------	----------	----------
ABOVE PRODUCT SUBTRACTED FROM ONE:		D(1)	D(2)	D(3)	D(4)

WORKSHEET No. 4

MODULE RELIABILITY CALCULATION - WORST CASE MODULE: --

$$ADC(i) = \frac{A(i)*C(i)}{1.00 - D(i)*[1.00 - A(i)]} = \frac{\text{substitute appropriate } A(i) \text{ and } C(i)}{\text{substitute appropriate } A(i) \text{ and } D(i)} = \frac{\text{calculate}}{\qquad\qquad}$$

$$ADC(1) = \frac{A(1)*C(1)}{1.00 - D(1)*[1.00 - A(1)]} = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = \_\_\_\_\_\_\_\_$$

$$ADC(2) = \frac{A(2)*C(2)}{1.00 - D(2)*[1.00 - A(2)]} = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = \_\_\_\_\_\_\_\_$$

$$ADC(3) = \frac{A(3)*C(3)}{1.00 - D(3)*[1.00 - A(3)]} = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = \_\_\_\_\_\_\_\_$$

$$ADC(4) = \frac{A(4)*C(4)}{1.00 - D(4)*[1.00 - A(4)]} = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = \_\_\_\_\_\_\_\_$$

SUM ALL THE ADC(i) TERMS --> ---------------

```
  ||
  ||
(copy)
  ||
  ||
 \||/
  \/
```

CALCULATED ENHANCEMENT FACTOR - WORST CASE ---------------

WORKSHEET No. 1

INHERENT CHARACTERISTICS - C(i) NOMINAL CASE MODULE: --

FACTOR	(check)	C(1)	C(2)	C(3)	C(4)
PREDOMINANTLY CONTROL	-------	.372	.299	.192	.138
PREDOMINANTLY REAL TIME	-------	.308	.308	.203	.166
PREDOMINANTLY INPUT/OUTPUT	-------	.205	.256	.423	.105
PREDOMINANTLY INTERACTIVE	-------	.270	.342	.264	.121
PREDOMINANTLY COMPUTATIONAL	-------	.277	.169	.133	.411
MANY DISTINCT OPERATIONAL MISSIONS	-------	.385	.299	.175	.141
SEVERAL VARIATIONS OF OPERATIONAL MISSIONS	-------	.362	.298	.178	.162
SINGLE OPERATIONAL MISSION	-------	.323	.251	.194	.212
MANY OPERATIONS REQUIRED - HIGHLY INTERRELATED	-------	.385	.316	.151	.149
MANY OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	-------	.372	.236	.187	.191
FEW OPERATIONS REQUIRED - HIGHLY INTERRELATED	-------	.346	.316	.168	.172
FEW OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	-------	.343	.240	.182	.218
EXTENSIVE HARDWARE INTERFACE REQUIREMENTS	-------	.262	.425	.209	.104
MINIMAL HARDWARE INTERFACE REQUIREMENTS	-------	.311	.284	.214	.175
EXTENSIVE SOFTWARE INTERFACE REQUIREMENTS	-------	.284	.417	.175	.113
MINIMAL SOFTWARE INTERFACE REQUIREMENTS	-------	.307	.275	.215	.189
EXTENSIVE HUMAN INTERFACE REQUIREMENTS	-------	.270	.341	.267	.112
MINIMAL HUMAN INTERFACE REQUIREMENTS	-------	.322	.269	.213	.178
WIDE RANGE OF ERROR-PRONE INPUTS	-------	.303	.236	.283	.178
WIDE RANGE OF ERROR-FREE INPUTS	-------	.304	.253	.247	.192
NARROW RANGE OF ERROR-PRONE INPUTS	-------	.313	.236	.259	.180
NARROW RANGE OF ERROR-FREE INPUTS	-------	.311	.237	.248	.200

SUM OF THE VALUES CHECKED: ---------- ---------- ---------- ----------

ABOVE SUM DIVIDED BY THE NUMBER OF CHECKS: ---------- ---------- ---------- ----------

C(1) C(2) C(3) C(4)

WORKSHEET No. 2

ERROR AVOIDANCE TECHNIQUES - A(i) NOMINAL CASE MODULE: --

FACTOR	(check)	1-A(1)	1-A(2)	1-A(3)	1-A(4)
INDEPENDENT QUALITY ASSURANCE ORGANIZATION	-------	.646	.653	.692	.697
INDEPENDENT TEST ORGANIZATION	-------	.696	.662	.651	.646
INDEPENDENT VERIFICATION AND VALIDATION (IV&V)	-------	.631	.661	.648	.622
USE OF A SOFTWARE SUPPORT LIBRARY	-------	.779	.757	.772	.757
USE OF A SOFTWARE CONFIGURATION CONTROL BOARD	-------	.775	.679	.777	.831
THOROUGH AND ENFORCED SOFTWARE DEVELOPMENT PLAN	-------	.672	.652	.668	.678
RIGIDLY CONTROLLED SYSTEM REQUIREMENTS SPEC	-------	.646	.617	.677	.698
RIGIDLY CONTROLLED INTERFACE DESIGN SPEC	-------	.698	.466	.630	.769
RIGIDLY CONTROLLED SOFTWARE REQUIREMENTS SPEC	-------	.598	.605	.634	.660
RIGIDLY CONTROLLED SOFTWARE FUNCTIONAL DESIGN SPEC	-------	.637	.612	.655	.656
RIGIDLY CONTROLLED SOFTWARE DETAILED DESIGN SPEC	-------	.617	.608	.610	.597
REQUIREMENTS TRACEABILITY MATRIX	-------	.645	.681	.738	.771
STRUCTURED ANALYSIS TOOLS	-------	.663	.694	.717	.742
PROGRAM SPECIFICATION LANGUAGE (PSL)	-------	.682	.710	.748	.752
PROGRAM DESIGN LANGUAGE (PDL)	-------	.643	.687	.721	.714
HIGH ORDER LANGUAGE (HOL)	-------	.674	.706	.698	.651
HIERARCHICAL, TOP-DOWN DESIGN	-------	.617	.633	.685	.735
STRUCTURED DESIGN	-------	.619	.640	.677	.715
SINGLE FUNCTION MODULARIZATION	-------	.625	.674	.698	.692
STRUCTURED CODE	-------	.629	.695	.720	.663
USE OF AUTOMATIC MEASUREMENT TOOLS	-------	.761	.780	.772	.755
USE OF AUTOMATIC TEST TOOLS	-------	.699	.718	.713	.692
PRODUCT OF THE VALUES CHECKED:		----------	----------	----------	----------
ABOVE PRODUCT SUBTRACTED FROM ONE:		----------	----------	----------	----------
		A(1)	A(2)	A(3)	A(4)

WORKSHEET No. 3

ERROR DETECTION TECHNIQUES - D(i) NOMINAL CASE MODULE: ---

FACTOR	(check)	1-D(1)	1-D(2)	1-D(3)	1-D(4)
FREQUENT PEER WALKTHROUGHS	-------	.548	.595	.621	.607
INFREQUENT PEER WALKTHROUGHS	-------	.749	.775	.805	.797
FREQUENT PROGRESS REVIEWS	-------	.755	.780	.785	.802
INFREQUENT PROGRESS REVIEWS	-------	.890	.890	.898	.912
FREQUENT QUALITY AUDITS	-------	.712	.731	.733	.755
INFREQUENT QUALITY AUDITS	-------	.862	.868	.880	.882
USE OF SOFTWARE PROBLEM REPORTS PRIOR TO PQT	-------	.668	.668	.703	.685
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO PQT	-------	.735	.722	.748	.742
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO FQT	-------	.755	.716	.763	.754
USE OF SPECIFICATION CHANGE NOTICES (SCN's)	-------	.807	.779	.811	.829
USE OF ENGINEERING CHANGE NOTICES (ECN's)	-------	.799	.769	.802	.818
SOFTWARE REQUIREMENTS REVIEW (SRR)	-------	.686	.669	.720	.766
PRELIMINARY DESIGN REVIEW (PDR)	-------	.714	.679	.721	.773
CRITICAL DESIGN REVIEW (CDR)	-------	.677	.672	.695	.738
TEST READINESS REVIEW (TRR)	-------	.799	.766	.781	.797
FUNCTIONAL CONFIGURATION AUDIT (FCA)	-------	.818	.782	.792	.817
PHYSICAL CONFIGURATION AUDIT (PCA)	-------	.851	.826	.823	.855
INFORMAL UNIT-LEVEL TESTING	-------	.566	.682	.655	.544
PRELIMINARY QUALIFICATION TESTING (PQT)	-------	.639	.659	.682	.679
FORMAL QUALIFICATION TESTING (FQT)	-------	.686	.700	.711	.698
SOFTWARE INTEGRATION TESTING	-------	.651	.550	.618	.666
SYSTEM INTEGRATION TESTING	-------	.670	.587	.633	.688
OPERATIONAL FIELD TESTING	-------	.673	.648	.637	.689
	PRODUCT OF THE VALUES CHECKED:	----------	----------	----------	----------
	ABOVE PRODUCT SUBTRACTED FROM ONE:	----------	----------	----------	----------
		D(1)	D(2)	D(3)	D(4)

WORKSHEET No. 4

MODULE RELIABILITY CALCULATION - NOMINAL CASE   MODULE: --

$$ADC(i) = \frac{A(i)*C(i)}{1.00 - D(i)*[1.00 - A(i)]} = \frac{\text{substitute appropriate } A(i) \text{ and } C(i)}{\text{substitute appropriate } A(i) \text{ and } D(i)} = \frac{\text{calculate}}{}$$

$$ADC(1) = \frac{A(1)*C(1)}{1.00 - D(1)*[1.00 - A(1)]} = \text{---------------} = \text{--------}$$

$$ADC(2) = \frac{A(2)*C(2)}{1.00 - D(2)*[1.00 - A(2)]} = \text{---------------} = \text{--------}$$

$$ADC(3) = \frac{A(3)*C(3)}{1.00 - D(3)*[1.00 - A(3)]} = \text{---------------} = \text{--------}$$

$$ADC(4) = \frac{A(4)*C(4)}{1.00 - D(4)*[1.00 - A(4)]} = \text{---------------} = \text{--------}$$

SUM ALL THE ADC(i) TERMS --> ---------------

(COPY)

I-11 CALCULATED ENHANCEMENT FACTOR - NOMINAL CASE ---------------

WORKSHEET No. 1

INHERENT CHARACTERISTICS - C(i) BEST CASE MODULE: --

FACTOR	(check)	C(1)	C(2)	C(3)	C(4)
PREDOMINANTLY CONTROL	-------	.372	.299	.192	.136
PREDOMINANTLY REAL TIME	-------	.308	.308	.203	.166
PREDOMINANTLY INPUT/OUTPUT	-------	.205	.256	.423	.105
PREDOMINANTLY INTERACTIVE	-------	.270	.342	.264	.121
PREDOMINANTLY COMPUTATIONAL	-------	.277	.169	.133	.411
MANY DISTINCT OPERATIONAL MISSIONS	-------	.385	.299	.175	.141
SEVERAL VARIATIONS OF OPERATIONAL MISSIONS	-------	.362	.298	.178	.162
SINGLE OPERATIONAL MISSION	-------	.323	.251	.194	.212
MANY OPERATIONS REQUIRED - HIGHLY INTERRELATED	-------	.385	.316	.151	.149
MANY OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	-------	.372	.236	.187	.191
FEW OPERATIONS REQUIRED - HIGHLY INTERRELATED	-------	.346	.316	.168	.172
FEW OPERATIONS REQUIRED - RELATIVELY INDEPENDENT	-------	.343	.240	.182	.218
EXTENSIVE HARDWARE INTERFACE REQUIREMENTS	-------	.262	.425	.209	.104
MINIMAL HARDWARE INTERFACE REQUIREMENTS	-------	.311	.284	.214	.175
EXTENSIVE SOFTWARE INTERFACE REQUIREMENTS	-------	.284	.417	.175	.113
MINIMAL SOFTWARE INTERFACE REQUIREMENTS	-------	.307	.275	.215	.189
EXTENSIVE HUMAN INTERFACE REQUIREMENTS	-------	.270	.341	.267	.112
MINIMAL HUMAN INTERFACE REQUIREMENTS	-------	.322	.269	.213	.176
WIDE RANGE OF ERROR-PRONE INPUTS	-------	.303	.236	.283	.178
WIDE RANGE OF ERROR-FREE INPUTS	-------	.304	.253	.247	.192
NARROW RANGE OF ERROR-PRONE INPUTS	-------	.313	.236	.259	.180
NARROW RANGE OF ERROR-FREE INPUTS	-------	.311	.237	.248	.200
SUM OF THE VALUES CHECKED:		----------	----------	----------	----------
ABOVE SUM DIVIDED BY THE NUMBER OF CHECKS:		---------- C(1)	---------- C(2)	---------- C(3)	---------- C(4)

WORKSHEET No. 2

ERROR AVOIDANCE TECHNIQUES - A(i) BEST CASE MODULE: --

FACTOR	(check)	1-A(1)	1-A(2)	1-A(3)	1-A(4)
INDEPENDENT QUALITY ASSURANCE ORGANIZATION	-------	.588	.600	.637	.639
INDEPENDENT TEST ORGANIZATION	-------	.644	.609	.596	.594
INDEPENDENT VERIFICATION AND VALIDATION (IV&V)	-------	.578	.610	.593	.566
USE OF A SOFTWARE SUPPORT LIBRARY	-------	.737	.713	.730	.712
USE OF A SOFTWARE CONFIGURATION CONTROL BOARD	-------	.730	.629	.734	.792
THOROUGH AND ENFORCED SOFTWARE DEVELOPMENT PLAN	-------	.620	.602	.617	.625
RIGIDLY CONTROLLED SYSTEM REQUIREMENTS SPEC	-------	.597	.569	.631	.648
RIGIDLY CONTROLLED INTERFACE DESIGN SPEC	-------	.649	.417	.579	.722
RIGIDLY CONTROLLED SOFTWARE REQUIREMENTS SPEC	-------	.549	.557	.586	.614
RIGIDLY CONTROLLED SOFTWARE FUNCTIONAL DESIGN SPEC	-------	.591	.564	.607	.608
RIGIDLY CONTROLLED SOFTWARE DETAILED DESIGN SPEC	-------	.568	.556	.559	.546
REQUIREMENTS TRACEABILITY MATRIX	-------	.589	.634	.689	.724
STRUCTURED ANALYSIS TOOLS	-------	.612	.648	.669	.695
PROGRAM SPECIFICATION LANGUAGE (PSL)	-------	.632	.661	.703	.702
PROGRAM DESIGN LANGUAGE (PDL)	-------	.597	.642	.675	.666
HIGH ORDER LANGUAGE (HOL)	-------	.626	.659	.649	.603
HIERARCHICAL, TOP-DOWN DESIGN	-------	.570	.587	.637	.688
STRUCTURED DESIGN	-------	.573	.592	.628	.669
SINGLE FUNCTION MODULARIZATION	-------	.575	.623	.645	.642
STRUCTURED CODE	-------	.581	.649	.670	.613
USE OF AUTOMATIC MEASUREMENT TOOLS	-------	.717	.736	.728	.706
USE OF AUTOMATIC TEST TOOLS	-------	.650	.671	.665	.642

PRODUCT OF THE VALUES CHECKED:	----------	----------	----------	----------
ABOVE PRODUCT SUBTRACTED FROM ONE:	----------	----------	----------	----------
	A(1)	A(2)	A(3)	A(4)

WORKSHEET No. 3

ERROR DETECTION TECHNIQUES - D(i) TEST CASE MODULE: --

FACTOR	(check)	1-D(1)	1-D(2)	1-D(3)	1-D(4)
FREQUENT PEER WALKTHROUGHS	-------	.501	.546	.571	.557
INFREQUENT PEER WALKTHROUGHS	-------	.713	.736	.770	.763
FREQUENT PROGRESS REVIEWS	-------	.709	.736	.741	.761
INFREQUENT PROGRESS REVIEWS	-------	.867	.865	.875	.892
FREQUENT QUALITY AUDITS	-------	.666	.685	.687	.710
INFREQUENT QUALITY AUDITS	-------	.833	.840	.855	.856
USE OF SOFTWARE PROBLEM REPORTS PRIOR TO PQT	-------	.616	.617	.652	.630
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO PQT	-------	.651	.675	.703	.697
USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO FQT	-------	.706	.662	.717	.706
USE OF SPECIFICATION CHANGE NOTICES (SCN's)	-------	.760	.735	.772	.759
USE OF ENGINEERING CHANGE NOTICES (ECN's)	-------	.755	.723	.758	.771
SOFTWARE REQUIREMENTS REVIEW (SRR)	-------	.637	.614	.673	.721
PRELIMINARY DESIGN REVIEW (PDR)	-------	.672	.639	.680	.737
CRITICAL DESIGN REVIEW (CDR)	-------	.634	.632	.657	.696
TEST READINESS REVIEW (TRR)	-------	.760	.722	.741	.756
FUNCTIONAL CONFIGURATION AUDIT (FCA)	-------	.776	.739	.747	.775
PHYSICAL CONFIGURATION AUDIT (PCA)	-------	.813	.790	.786	.818
INFORMAL UNIT-LEVEL TESTING	-------	.519	.675	.603	.495
PRELIMINARY QUALIFICATION TESTING (PQT)	-------	.553	.610	.626	.631
FORMAL QUALIFICATION TESTING (FQT)	-------	.640	.656	.663	.648
SOFTWARE INTEGRATION TESTING	-------	.605	.605	.722	[illegible]
SYSTEM INTEGRATION TESTING	-------	.623	.641	.785	.626
OPERATIONAL FIELD TESTING	-------	.640	.616	.605	.654

PRODUCT OF THE VALUES CHECKED:	----------	----------	----------	----------
ABOVE PRODUCT SUBTRACTED FROM ONE:	---------- D(1)	---------- D(2)	---------- D(3)	---------- D(4)

WORKSHEET No. 4

MODULE RELIABILITY CALCULATION - BEST CASE MODULE: --

	A(i)*C(i)		substitute appropriate A(i) and C(i)		calculate
ADC(i) =	--------------------------	=	--	=	------------
	1.00 - D(i)*[1.00 - A(i)]		substitute appropriate A(i) and D(i)		
	A(1)*C(1)				
ADC(1) =	--------------------------	=	--	=	------------
	1.00 - D(1)*[1.00 - A(1)]				
	A(2)*C(2)				
ADC(2) =	--------------------------	=	--	=	------------
	1.00 - D(2)*[1.00 - A(2)]				
	A(3)*C(3)				
ADC(3) =	--------------------------	=	--	=	------------
	1.00 - D(3)*[1.00 - A(3)]				
	A(4)*C(4)				
ADC(4) =	--------------------------	=	--	=	------------
	1.00 - D(4)*[1.00 - A(4)]				

SUM ALL THE ADC(i) TERMS --> ------------

(COPY)

CALCULATED ENHANCEMENT FACTOR - BEST CASE ------------

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

DTIC
FILMED
4-86
END